IBM Content Manager OnDemand Web Enablement Kit Java APIs
The Basics and Beyond

Develop Web applications by using ODWEK V8.4 Java APIs

Gain insightful best practices, hints, and tips

Tune and troubleshoot OnDemand Web applications

Wei-Dong Zhu
Mark Mikeal
Hassan A. Shazly
Elliott Wade
Sebastian Welter

Redbooks

ibm.com/redbooks

International Technical Support Organization

IBM Content Manager OnDemand Web Enablement Kit Java APIs: The Basics and Beyond

December 2008

SG24-7646-00

Note: Before using this information and the product it supports, read the information in "Notices" on page ix.

First Edition (December 2008)

This edition applies to Version 8 Release 4 of IBM Content Manager OnDemand (program number 5724-J33).

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®
alphaWorks®
DB2®
i5/OS®
IBM®
OS/390®
OS/400®
POWER™
Rational®
Redbooks®
Redbooks (logo) ®
System i®
System z®
Tivoli®
WebSphere®
z/OS®

The following terms are trademarks of other companies:

Adobe Reader, Adobe, and Portable Document Format (PDF) are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Cell Broadband Engine and Cell/B.E. are trademarks of Sony Computer Entertainment, Inc., in the United States, other countries, or both and is used under license therefrom.

100% Pure Java, Java, Javadoc, JavaScript, JavaServer, JDK, JNI, JSP, JVM, Solaris, Sun, SunOS, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

ESP, Excel, Internet Explorer, Microsoft, Windows Vista, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel Pentium M, Intel Pentium, Intel, Pentium M, Pentium, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

IBM® Content Manager OnDemand is the industry leading report management product. It provides enterprise report management and electronic statement presentment. It is high-performance middleware for automatic management of formatted computer output and reports. Content Manager OnDemand helps companies gain significant return on investment by transforming costly high-volume print output to electronic information capture and presentation in support of customer service.

Content Manager OnDemand includes the OnDemand Web Enablement Kit (ODWEK) with which companies can build their own Web applications for interfacing with Content Manager OnDemand. Many companies have used ODWEK, and the usage continues to grow significantly as companies integrate Content Manager OnDemand into their enterprise Web and portal solutions.

In this IBM Redbooks® publication, we provide an overview of the ODWEK version 8.4 Java™ APIs and explain the commonly used APIs for application development. In addition, we examine the capabilities and usage of the APIs through use cases, best practices, hints and tips, and code snippets. We also explain connection pooling, folder searching, document retrieval, document storing and updating, memory and performance, and troubleshooting in terms of application development.

The ODWEK Java APIs can be incorporated into any Java-based application, including stand-alone applications, portlets, servlets and Web services. We illustrate the APIs by using servlet-based code to encourage an integrated understanding of the topics.

This book is intended for application developers who are responsible for developing Web applications that interface with Content Manager OnDemand. It also serves as a good reference guide for developers and system administrators to fine-tune and troubleshoot Content Manager OnDemand Web applications.

The team that wrote this book

This book was developed in Boulder, Colorado by a team of specialists from around the world working with the IBM International Technical Support Organization (ITSO).

Wei-Dong (Jackie) Zhu is an ITSO project leader specializing in enterprise content management, risk, and discovery management. She has more than 10 years of software development experience in accounting, image workflow processing, and digital media distribution. Jackie joined IBM in 1996. She is a Certified Solution Designer for IBM Content Manager and has managed and lead the production of many enterprise content management, risk, and discovery management Redbooks publications. Jackie holds a Master of Science degree in Computer Science from the University of Southern California.

Mark Mikeal is a senior managing consultant team leader with IBM Enterprise Content Management Lab Services. He has more than 25 years of software development and integration experience in enterprise content management. Mark joined IBM in 1984 and is a Certified Specialist for cross-industry solutions related to enterprise content management. He has presented at multiple industry conferences. Mark holds a Bachelor of Science degree in Business Administration from the University of South Carolina.

Hassan (Al) A. Shazly is a senior software engineer with Enterprise Content Management OnDemand and has been with IBM since 1996. He has over 30 years of software management and development experience in various business and scientific applications. He has been instrumental in the design, development, and product testing of the OnDemand Content Management system. Hassan is certified in both Content Manager OnDemand and On Demand Business. He has over 20 publications and has presented at multiple technical conferences. Hassan holds a Ph.D. in Remote Sensing and Image Processing from the University of South Carolina.

Elliott Wade is a software engineer and architect with more than 10 years of Java development experience. He has worked as a technology consultant in the financial services and manufacturing sectors and is currently a senior developer at ADP Retirement Services. His Content Manager OnDemand projects have included stand-alone applications, batch processes, and middle-tier integration. Elliott studied Computer Science at Boston University.

Sebastian Welter is a technical sales consultant with Enterprise Content Management software in Germany. He has about four years of software management and development experience in various business applications. Sebastian joined IBM in 2004 and has worked on numerous Content Manager OnDemand assets. Sebastian holds a diploma in Business Information Systems from the University of Corporate Education in Mannheim and a Bachelor of Arts degree in Business Administration.

Many thanks to the following people who have shared their knowledge and contributed substantial material for this book:

- *Gordon Campbell*, Certified Software IT Specialist, for providing the base sample API programs and API description documentation
- *Nelson Chen*, Content Manager OnDemand Software Developer, for providing education and globalization documentation
- *Brian Hoyt*, Content Manager OnDemand Software Architect, for providing programming insight, information, and review
- *Hubert Hwang*, Content Manager OnDemand Level 2 Support, for providing valuable troubleshooting hints and tips documentation
- *Bob Lichens*, Content Manager OnDemand Software Developer, for providing guidance and documentation on best practices, hints, and tips

We also thank the following people for their contributions to this project:

Stephen Henrikson
Thomas Garcia
Gregory Felderman
Eric Hann
Ben Boltz
Janice L. Holoman
Kevin Van Winkle
IBM Software Group, Content Manager OnDemand, Boulder, Colorado

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Part 1

Programming with the ODWEK Java APIs

In this part, we provide an overview ODWEK Java API overview and describe the basic API usage with examples. In this part, we include the following chapters:

1

ODWEK Java API overview

To effectively use the OnDemand Web Enablement Kit (ODWEK) Java APIs, you must understand the architecture of both the IBM Content Manager OnDemand (OnDemand) server and the ODWEK Java APIs. In this chapter, we provide an overview of both and introduce fundamental topics that are covered in detail in this book.

This chapter covers the following topics:

- Origin of the ODWEK Java APIs
- Content Manager OnDemand system overview
- ODWEK Java APIs function overview
- ODWEK Java APIs architecture
- ODWEK Java APIs components
- WEBi client

1.1 Origin of the ODWEK Java APIs

Content Manager OnDemand provides report management capabilities including the capturing and archiving of computer output, such as printed reports, and the searching and retrieving of stored content. ODWEK is a toolkit that is provided with Content Manager OnDemand to enable access and management of the OnDemand server and its data. The ODWEK Java APIs are part of the toolkit that you can use to write applications to access and manage data stored in OnDemand servers.

ODWEK began as an IBM Services offering, called the *OnDemand Internet Client*. OnDemand Internet Client evolved into an IBM project called the *OnDemand Web Enablement Kit*, which was released with version 2.1.1.9 of Content Manager OnDemand. As of this writing, the current release of ODWEK is version 8.4.0.2. Over the years, many functions and features have been added. The Java APIs were introduced with release 7.1.0.2 of ODWEK.

Before we provide an overview of the ODWEK Java APIs, you must have an overall understanding of Content Manager OnDemand system, which we provide briefly in the next section. If you are already familiar with Content Manager OnDemand, skip the next section and proceed to 1.3, “ODWEK Java APIs function overview” on page 11.

1.2 Content Manager OnDemand system overview

A Content Manager OnDemand system processes computer output, extracts index information from data, stores the index information in a relational database, and stores the data in the Content Manager OnDemand archive. It also enables users to search and retrieve the stored data in the Content Manager OnDemand archive.

In this section, we provide an overview of the Content Manager OnDemand system architecture. It introduces Content Manager OnDemand data, its data model, and discusses data indexing and loading.

1.2.1 Content Manager OnDemand Library Server and Object Servers

In general, Content Manager OnDemand server functionality is identical across all supported platforms. The Content Manager OnDemand server environment includes a Library Server and one or more Object Servers that reside on systems connected to a TCP/IP network. Figure 1-1 on page 5 illustrates this arrangement.

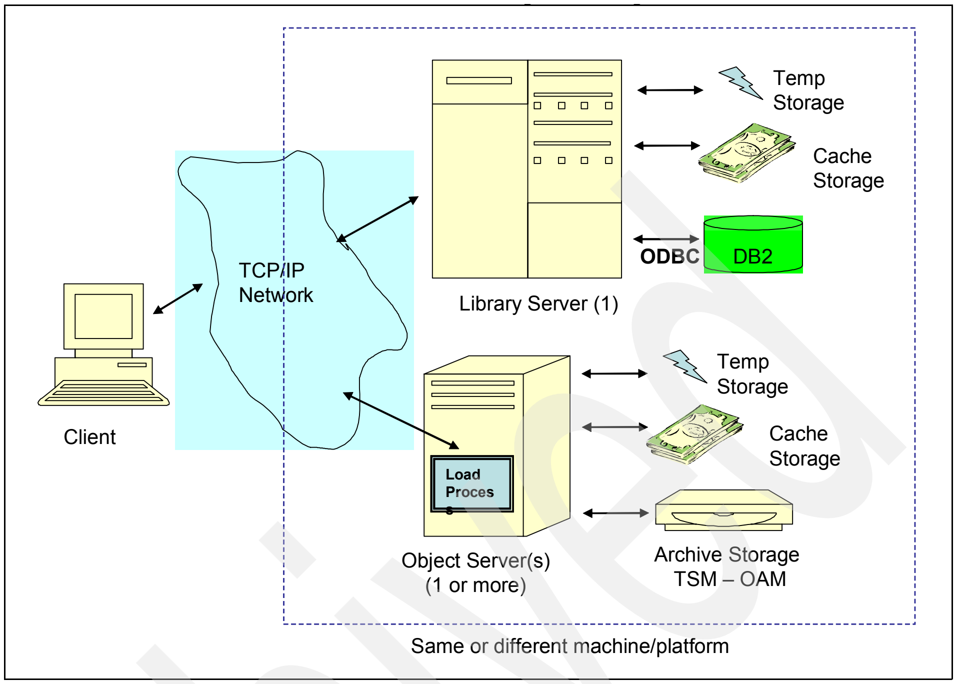


Figure 1-1   System overview: Library Server and Object Servers

Library Server

The *Library Server* maintains a central database that contains an index of the reports and documents that are stored in Content Manager OnDemand. The Library Server database also contains administrative objects such as users, groups, printers, application groups, applications, folders, cabinets, and storage sets. The Library Server processes client logons, queries, print requests, and updates to the database.

In addition to the database itself, the Library Server consists of a set of managers, which include the request manager, the database manager, and the server print manager. The request manager receives, and routes or responds to, incoming requests. The database manager provides the database engine and database administration utilities. The server print manager schedules and manages print requests.

Object Server

An *Object Server* maintains documents on persistent storage volumes. A *storage volume* is either a cache or an archive volume. A *cache volume* is typically disk-based and is intended to store frequently-accessed documents. Many Content Manager OnDemand installations implement only this type of storage.

Archive storage and the associated archive storage manager can be configured to access data that are for long-term storage and might be needed less frequently from media library or other archival storage systems. Archive storage is optional.

An Object Server loads data, retrieves documents, and expires documents. Data loading and maintenance utilities are installed on Object Servers.

Logical server

The combination of a single Library Server and multiple Object Servers is regarded as a *logical server*. Any of this server's components can run on a supported operating system.

In this book, when we refer to an OnDemand server or a system without indicating a particular subsystem, we are referring to the logical server. Although this logical server, which is also considered as a logical system, is treated as a unit, the number of physical servers and administrative roles depends on particulars of the architecture and organization.

1.2.2 Content Manager OnDemand data

The Content Manager OnDemand data consists of reports and documents.

Reports and documents

A *report* represents the data loaded into Content Manager OnDemand in a single operation. It is a body of content that can be unloaded or archived.

A *document* represents a logical segment or portion of a report that can be requested and viewed by users who do not want (or are not permitted) to see the entire report. An indexing process divides reports into logical documents.

As an example, consider a county agency that produces a billing report and a transaction report, both on a monthly basis. To capture and retain the output of these two reports, the Content Manager OnDemand administrator creates an application called BILLS for the billing reports and an application called TRANS for the transaction report.

The *billing report* contains the property tax billing statement for every property owner in the county. The BILLS application uses the document indexing method to divide the report into documents. Each property owner's billing statement in the report becomes a *document* in Content Manager OnDemand. A user can retrieve a billing statement with a specific date. Content Manager OnDemand retrieves the corresponding document based on the property owner or property number, and the user specified date.

The *transaction report* includes all the payment transactions by the property owners. The TRANS application uses the report indexing method to divide the report into documents. A group of 100 pages in the report becomes a document. Each group is indexed by using the first and last sorted transaction values that occur in the group. A user can retrieve a group of pages that contains a specific transaction number by specifying the date and the transaction number. Content Manager OnDemand retrieves the corresponding document that contains the value that is entered by the user.

A document is a single object that is stored in Content Manager OnDemand. Examples of documents are word processing documents, TIF images, spreadsheets, and other type of documents or portions of reports.

1.2.3 Content Manager OnDemand data model

The Content Manager OnDemand data model is composed of applications, application groups, folders, and cabinets. Figure 1-2 on page 8 shows a sample implementation that stores student and faculty information for a university.

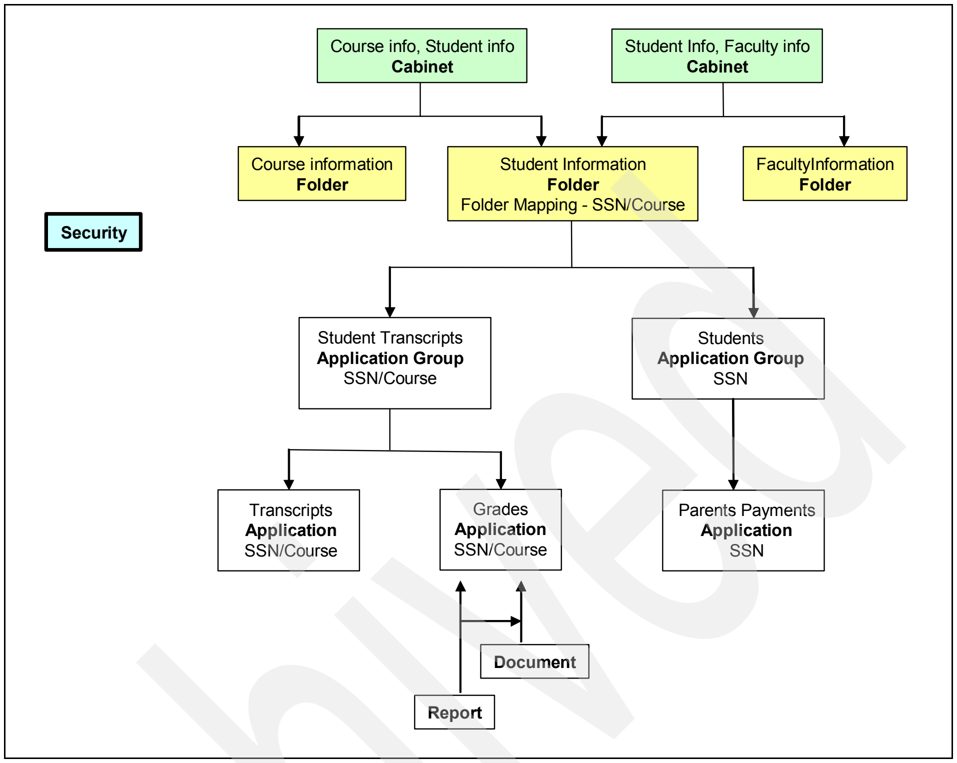


Figure 1-2 Example Content Manager OnDemand data model implementation

Application

An *application* describes the physical characteristics of a report or document to Content Manager OnDemand. Typically, you define an application for each report or document type to be stored in Content Manager OnDemand.

The application includes information about the format of the data, the orientation of the data on the page, the paper size, the record length, and the code page of the data. The application also includes parameters that the indexing program uses to locate and extract index data. In addition, it includes processing instructions that Content Manager OnDemand uses to load index data into the Library Server database and to load report and document data into the Content Manager OnDemand Object Server.

The student and faculty example in Figure 1-2 shows three applications, Transcripts, Grades, and Parents Payments, which generate student transcripts, grade reports, and payment information.

Application group

An *application group* is a collection of one or more applications with compatible indexing and storage management requirements. The application group defines the index information that is used to load, search for, and retrieve reports.

Multiple applications are combined into an application group so that users can access the documents in all of the grouped applications with a single query. The index keys of all applications in an application group must be compatible. That is, they must be of the same data type and length.

For the student and faculty example in Figure 1-2 on page 8, the Transcripts application uses student social security numbers (SSN) and course numbers (Course) as indexes for student transcripts. The Grades application also uses these two indexing fields for the grade reports. Therefore, these two applications can be grouped into one application group, called *Student Transcripts*. The Parents Payments application uses only the SSN as the index for the report. This application belongs to a separate application group, called *Students*.

Folder

A *folder* contains one or more application groups. A folder provides users with a convenient way to find related information that is stored in Content Manager OnDemand, regardless of the source of the information or how the data was prepared. By using folders, an administrator can set up a common query panel for several application groups that might use different indexing schemes. A user can retrieve the data with a single query.

In the student and faculty example in Figure 1-2 on page 8, a Student Information folder contains student transcripts, grades, and payment information. This information is stored in different application groups (Student Transcripts and Students) that are defined in different applications (Transcripts, Grades, Parents Payments) and are created by different programs. When a folder search is performed, Content Manager OnDemand searches within each application group that is part of the folder.

Cabinet

A *cabinet* is a container for one or more folders. You can identify a group of folders as belonging to a cabinet for convenient navigation. A folder can belong to multiple cabinets.

The student and faculty example in Figure 1-2 on page 8 shows one cabinet that contains both the Course Information and Student Information folders and another cabinet that contains both the Student Information and the Faculty Information folders. Therefore, with the first cabinet, a user can locate folders that contain information that is related to both students and course information. For the second cabinet, a user can locate folders that contain information that is related to students and faculty. Note that cabinets do not enable searching. They only group folders for easy access.

1.2.4 Content Manager OnDemand data indexing and loading

User applications generate printed output that can be saved as report data on disk (on one of the Object Servers). The OnDemand server indexes, segments, compresses and loads the report data into the OnDemand server. The indexes are stored in the Content Manager OnDemand Library Server. The compressed documents are stored in one of the Object Servers.

Indexing

The reports that are stored in Content Manager OnDemand must be indexed. Library Server index entries for a particular report provide users an efficient means to request and view specific report data. Content Manager OnDemand supports several types of index data and indexing programs (also known as *indexers*).

An administrator defines the index files and other processing parameters that are used by the selected indexer to locate and extract index information from reports. Data types that can be indexed and loaded include line data, AFP, TIF, PDF, and others.

TIF data: Technically, Content Manager OnDemand does not index TIF data. Content Manager OnDemand loads TIF data through the generic indexer, which does not index data, but rather gets the indexes from a specified file.

With Content Manager OnDemand, data can be indexed externally prior to being used in the system.

Loading

The Content Manager OnDemand data loading programs read the index data and load it into the Content Manager OnDemand database. The data loading programs obtain processing parameters from the Content Manager OnDemand database. Sample parameters include those that are used to segment, compress, and store report data in cache storage or archive storage.

1.3 ODWEK Java APIs function overview

The ODWEK Java APIs enable access to the OnDemand server from any IP network, whether intranet or Internet. By using the APIs, you can write applications that can access data stored in Content Manager OnDemand.

The ODWEK Java APIs provide the following functions for users to perform:

- Log into an OnDemand server.
- List cabinets or folders.
- Search for a specific set of documents in a folder.
- Retrieve documents.
- View documents.
- Log off from the system.

In addition to the common functions, by using the ODWEK Java APIs, users can perform the following functions:

- Change the Content Manager OnDemand account password.
- Print documents.
- Add document annotations.
- Update documents.

Custom applications can support some or all of these functions. This book addresses programming to support these functions and other functions that are not listed here for a simplified introduction.

The ODWEK Java APIs can access OnDemand servers that are running on any supported platform, including IBM z/OS®, System i®, AIX®, Linux® on System z, and Microsoft® Windows®.

1.3.1 Typical Web applications developed by using the ODWEK Java APIs

For Web applications, two broad scenarios exist for client access to data stored in Content Manager OnDemand. The first scenario is an *intranet* access scenario, in which users on the corporate network access the OnDemand server by using an application that connects directly to Content Manager OnDemand. Intranet applications usually expose a query-criteria interface to users, who typically have their own security profiles on the OnDemand server.

The second scenario is an *Internet* client-access scenario, in which external users are given access to a limited subset of the data stored in Content Manager OnDemand. The access is given usually through a middle-tier application such as a custom servlet or a Web service. An example is a banking application that allows customers to log on and view any of their statements for the previous twelve months. In this scenario, users are restricted to access their own documents, and thousands of users might access the system concurrently.

Application development techniques for both of the scenarios are discussed in detail in the following chapters.

Figure 1-3 shows a workstation with a Web browser that uses the ODWEK Java APIs to access data from an OnDemand server.

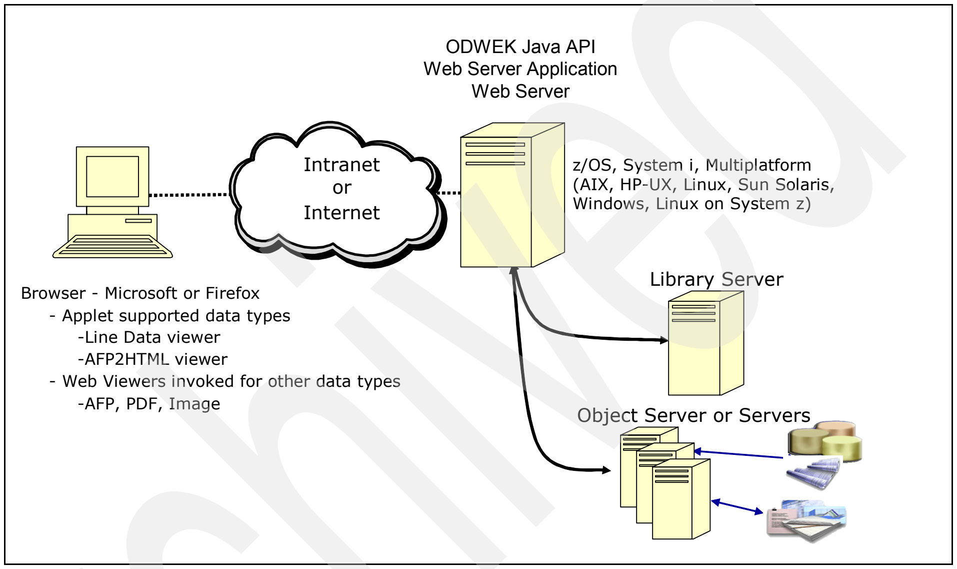

ODWEK Java API
Web Server Application
Web Server
Intranet or Internet
z/OS, System i, Multiplatform (AIX, HP-UX, Linux, Sun Solaris, Windows, Linux on System z)
Library Server
Object Server or Servers
Browser - Microsoft or Firefox
- Applet supported data types
-Line Data viewer
-AFP2HTML viewer
- Web Viewers invoked for other data types
-AFP, PDF, Image

Figure 1-3 Client access using ODWEK

We discuss applets and Web viewers shown in Figure 1-3 in 1.5.2, “Viewers” on page 14.

The architecture of Content Manager OnDemand enables systems that employ it to scale with usage to the limits of available hardware and network resources.

1.4 ODWEK Java APIs architecture

The ODWEK Java APIs provide a convenient programming interface for application developers. They allow architects and developers to choose the implementation that best suits their requirements. Using the ODWEK Java APIs is the preferred method for developing clients that access the OnDemand server.

You can use the ODWEK Java APIs to develop Web applications and to develop both batch and stand-alone applications. In addition, you can use the ODWEK Java APIs to create middleware that integrates existing applications with Content Manager OnDemand.

Figure 1-4 illustrates the common architectural modes of the ODWEK Java APIs.

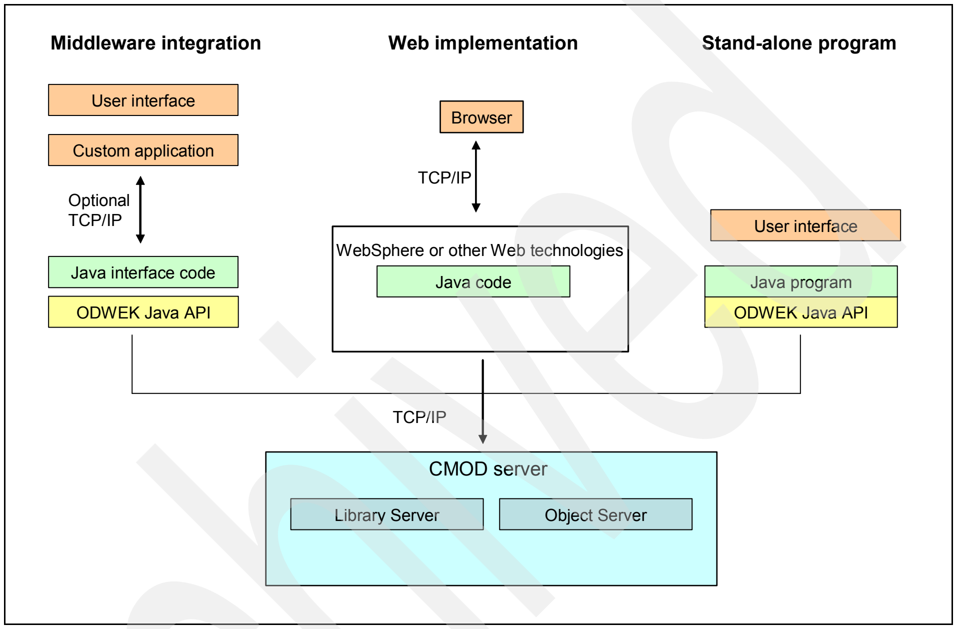


Figure 1-4 ODWEK-enabled application architectures

1.5 ODWEK Java APIs components

The ODWEK Java APIs comprise the standard libraries that you can use to write applications that access data stored in Content Manager OnDemand and viewers.

Note: This book is written based on ODWEK version 8.4.0.2.

1.5.1 ODWEK Java and native shared library

The API classes are in the com.ibm.edms.od Java package. These classes provide an object-oriented interface to the native Content Manager OnDemand libraries, which access OnDemand servers over the TCP/IP network.

For a list of distributed files, see 1.5.3, “ODWEK Java API distribution files” on page 15.

1.5.2 Viewers

Most ODWEK applications are capable of displaying retrieved documents or report segments. ODWEK provides several viewers that can display various document types in a browser window. Each viewer adds a toolbar to its display window, providing controls that help users to work with the retrieved documents.

A viewer provided with ODWEK is implemented as either a *browser plug-in* or an *applet*. Plug-ins must be installed locally on users’ workstations, where applets are dynamically downloaded according to the type of data to be viewed.

ODWEK provides the following viewers:

- Locally installed plug-ins
  - AFP Web Viewer plug-in
  - Image Web Viewer plug-in
- Dynamically downloaded applets
  - Java Line Data Viewer applet
  - Java AFP2HTML Viewer applet

An advantage of applets is that they are dynamically downloaded to users’ workstations. On the contrary, locally installed plug-ins require special software distribution. For example, when a new version of the AFP Web Viewer or Image Web Viewer plug-in is available, the new software must be distributed to users in order for them to benefit from the update.

The viewers that are provided with ODWEK retrieve documents from an OnDemand server in compressed forms. A viewer uncompresses the documents before displaying them in the browser. When a document is stored in Content Manager OnDemand as a *large object*, the viewer retrieves and uncompresses segments of the document as they are needed. That is, as the user scrolls through the document’s pages.

Viewers other than those provided with ODWEK can be integrated with the ODWEK Java APIs. Such viewers can support additional document types or provide custom document-related functions.

AFP Web Viewer plug-in

Browser plug-ins can extend the types of documents or media that a Web browser is capable of displaying. The AFP Web Viewer plug-in enables browsers with capabilities to search, retrieve, view, navigate, and print AFP documents.

Image Web Viewer plug-in

The Image Web Viewer plug-in enables browsers with capabilities to search, retrieve, view, navigate, and print documents in the BMP, GIF, JPEG, PCX, PNG, and TIFF formats.

Java Line Data Viewer applet

The Java Line Data Viewer applet displays line data documents in the browser window and provides toolbar controls that help users to work with the documents. An administrator enables the Java Line Data Viewer by using the ODConfig object.

Java AFP2HTML Viewer applet

The Java AFP2HTML Viewer is an applet with which users can view the output generated by the IBM AFP2WEB Transform service offering. AFP2WEB Transform converts AFP documents and resources into HTML documents. The Java AFP2HTML Viewer toolbar includes controls for working with large objects. After installing and configuring the AFP2WEB Transform, an administrator enables the Java AFP2HTML Viewer by using the ODConfig object.

Continue with the next section to see a list of distributed files.

1.5.3 ODWEK Java API distribution files

The ODWEK distribution includes the Java APIs, Common Gateway Interface (CGI), servlet, viewers, and AFP2WEB Transform.

The following list of ODWEK Java API binary distribution files is provided to acquaint developers with the distribution as an aid to use ODWEK in development environments, application servers, or stand-alone deployments. This listing might change because the distributed modules are subject to change and vary slightly by platform and release level.

- ODWEK Java and native shared libraries
  - Java API
    - ODApi.jar, Java API library
    - ODApiDoc.zip, Java API documentation (javadoc)
  - Native sockets interface
    - arssck32.dll (Windows)
  - Native shared libraries

• ars3wapi32.dll	(Windows)
• libars3wapi32.a	(AIX 32-bit)
• libars3wapi64.a	(AIX 64-bit)
• libars3wapi32.sl	(HPUX 32-bit)
• libars3wapi64.sl	(HPUX 64-bit)
• libars3wapi32.so	(SunOS™, iLinux 32-bit)
• libars3wapi64.so	(SunOS, iLinux 64-bit)
• libars3wapi64.so	(Linux on System z® 64-bit)
• libars3wapi32	(z/OS 32-bit)
• libars3wapi64	(z/OS 64-bit)

  - Locale

    Locale files are normally deployed with a full ODWEK installation.

    - arscpcs.cfg, code page to code set mappings
    - *.dll, various locale-specific libraries
  - ICU support

    ICU support files must be placed in a directory that is part of the executing environment's search path, typically defined by the path environment variable.

• icudt36.dll	(Windows)
• icuin36.dll	(Windows)
• icule36.dll	(Windows) No longer needed

Note: The icule36.dll is included for compatibility with any back-level applications that may be running.

• iculx36.dll	(Windows) No longer needed
• icuuc36.dll	(Windows)
• icudt36l.dat	(Windows, Linux)
• icudt36b.dat	(AIX, SunOS, HPUX) Leave this file in the locale installation directory

- Viewers (and transforms)
  - Configuration files for transforms

• afp2html.ini	Configures the AFP2WEB Transform
• afp2pdf.ini	Configures the AFP2PDF Transform
• arsxenos.ini	Configures the XENOS Transform

  - Applets
    - ODLineDataViewer2.jar

      The Content Manager OnDemand Line Data Viewer applet
    - ODAfp2Html2.jar

      The Content Manager OnDemand AFP Transform applet
    - IEFix.js

      The JavaScript™ file that assists the applet launch
  - Plug-ins

    Plug-ins are distributed as InstallShield self-extracting installations. To install a plug-in, download the installation file to the target Windows XP, Windows 2000, Windows 2003, or Windows Vista® system, and execute the file. If a browser is running while the installation is in progress, then you must restart the browser before you can use the new plug-in.
    - afpplgin.exe

      Content Manager OnDemand AFP Web Viewer (Plugin/ActiveX); supports all languages, double-byte character set (DBCS)
    - afpplgin.zip

      Content Manager OnDemand AFP Web Viewer (Plugin/ActiveX); compressed format, supports all languages, DBCS
    - imgplgin.exe

      Content Manager OnDemand Image Web Viewer (Plugin/ActiveX); supports all languages

1.6 WEBi client

WEBi (pronounced “webby”) is a fully functional Web 2.0 Content Manager OnDemand client that uses the ODWEK Java APIs for interaction with Content Manager OnDemand. It is an example Web application that was developed by using the ODWEK Java APIs. WEBi is developed to meet broad customer requirements. Many customers use it as their default Content Manager OnDemand client application.

Figure 1-5 shows the technology components and architecture of the WEBi implementation. The Web 2.0 technologies employed by WEBi demonstrate a rich and highly functional ODWEK user interface.

It is beyond the scope of this book to discuss WEBi in details. The information provided here is for reference purposes only.

CM OO API: In Figure 1-5, CM OO API stands for *Content Manager Object Oriented API*.

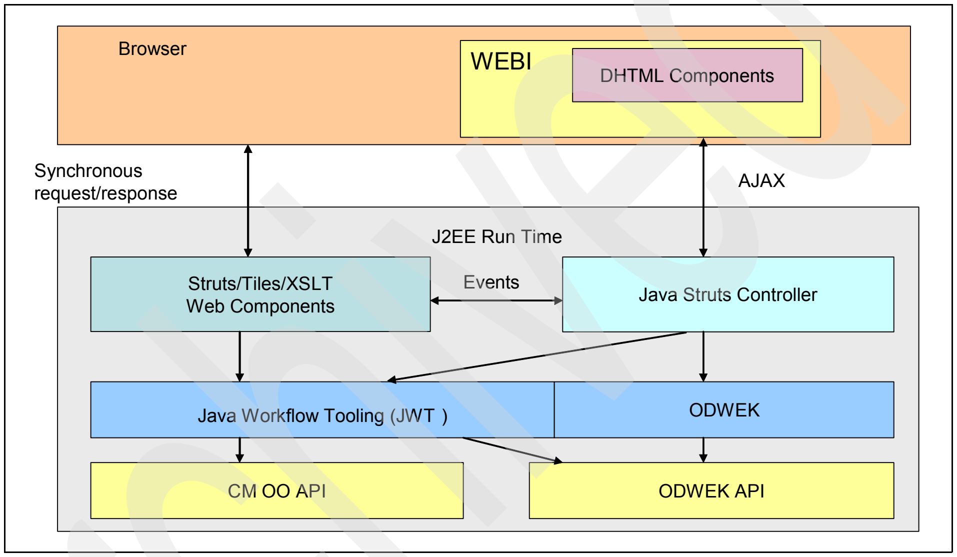
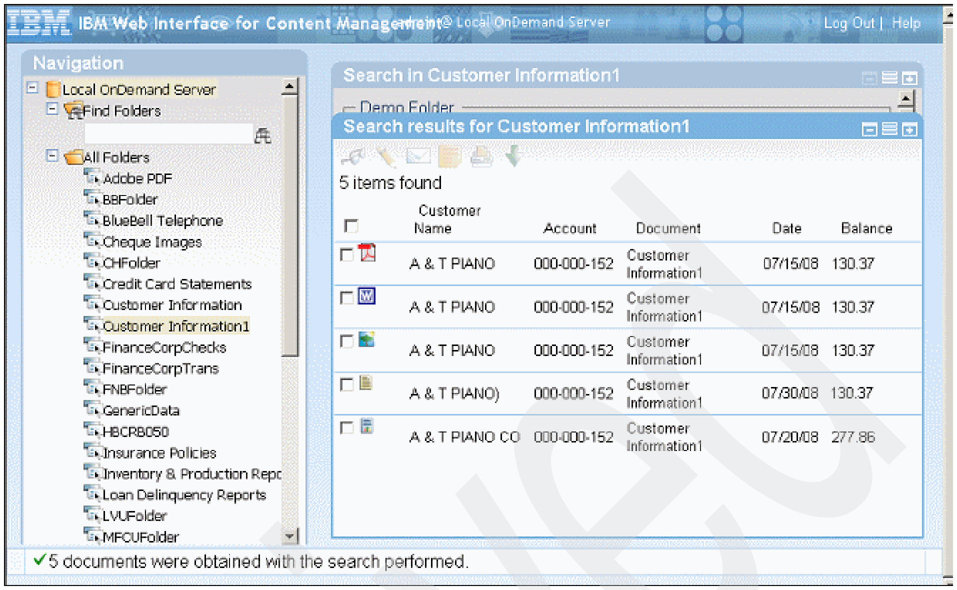


Figure 1-5 WEBI technology overview

The following windows show some of the WEBi user interface so that you can see the type of the application and user interface that you can create with the ODWEK Java APIs. Figure 1-6 on page 19 shows a WEBi user interface with a returned list of search result.


IBM Web Interface for Content Management® Local OnDemand Server
Log Out | Help
Navigation

- Local OnDemand Server
  - Find Folders
  - All Folders
    - Adobe PDF
    - BBFolder
    - BlueBell Telephone
    - Cheque Images
    - CHFolder
    - Credit Card Statements
    - Customer Information
    - Customer Information1
    - FinanceCorpChecks
    - FinanceCorpTrans
    - FNBFolder
    - GenericData
    - HBCRB050
    - Insurance Policies
    - Inventory & Production Repc
    - Loan Delinquency Reports
    - LVUFolder
    - MFCUFolder

Search in Customer Information1

Demo Folder

Search results for Customer Information1

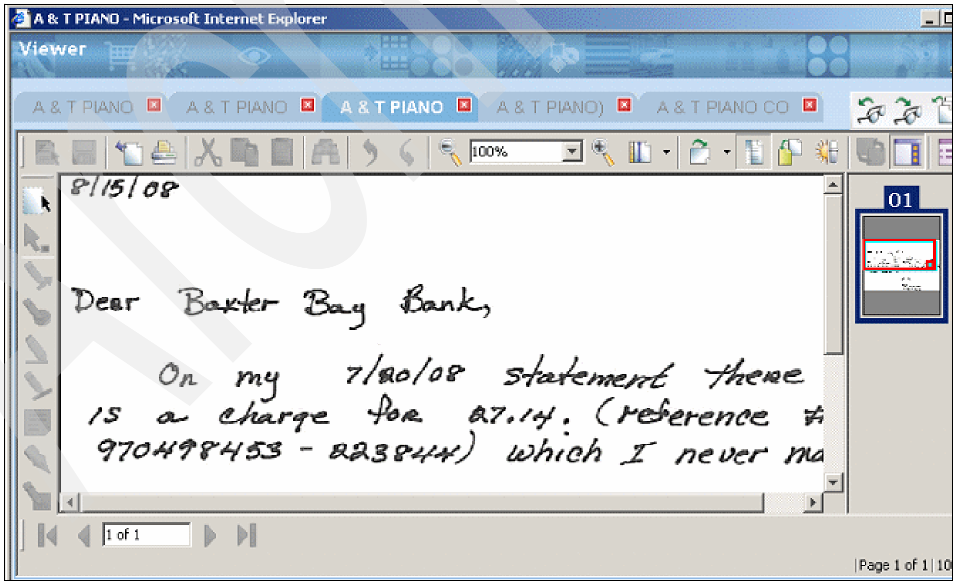
5 items found

Customer Name	Account	Document	Date	Balance
A & T PIANO	000-000-152	Customer Information1	07/15/08	130.37
A & T PIANO	000-000-152	Customer Information1	07/15/08	130.37
A & T PIANO	000-000-152	Customer Information1	07/15/08	130.37
A & T PIANO)	000-000-152	Customer Information1	07/30/08	130.37
A & T PIANO CO	000-000-152	Customer Information1	07/20/08	277.86

✓5 documents were obtained with the search performed.

Figure 1-6 WEBi user interface

Figure 1-7 shows a sample WEBi window with a retrieved hand-written customer inquiry.



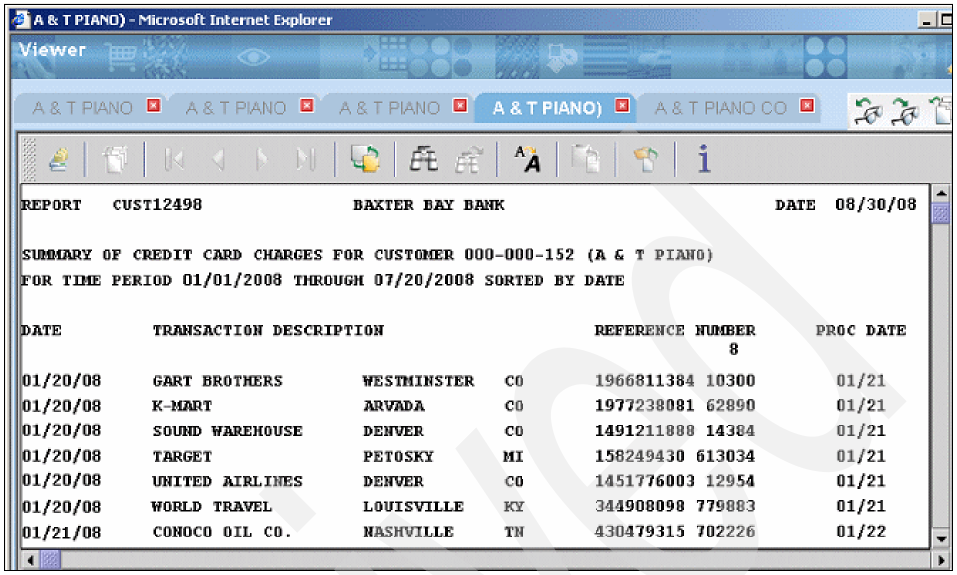
8/15/08

Dear Baxter Bay Bank,

On my 7/20/08 statement there is a charge for 27.14. (reference # 970498453 - 223844) which I never ma

Figure 1-7 Sample WEBi window showing a retrieved hand-written customer inquiry

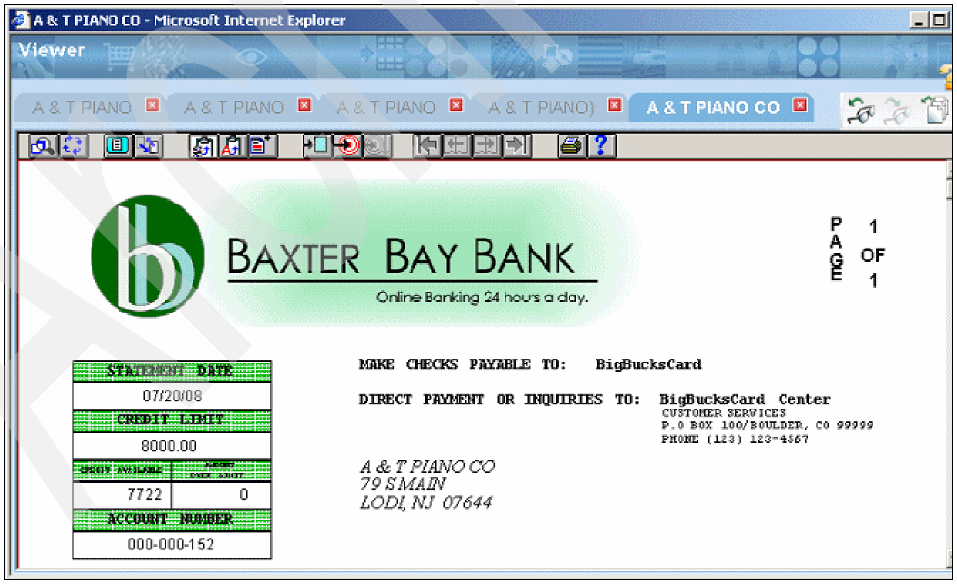
Figure 1-8 shows a sample WEBi window with a retrieved customer statement.



```
REPORT   CUST12498                BAXTER BAY BANK                        DATE  08/30/08

SUMMARY OF CREDIT CARD CHARGES FOR CUSTOMER 000-000-152 (A & T PIANO)
FOR TIME PERIOD 01/01/2008 THROUGH 07/20/2008 SORTED BY DATE

DATE         TRANSACTION DESCRIPTION                      REFERENCE NUMBER      PROC DATE
                                                                 8

01/20/08     GART BROTHERS        WESTMINSTER    CO       1966811384 10300        01/21
01/20/08     K-MART               ARVADA         CO       1977238081 62890        01/21
01/20/08     SOUND WAREHOUSE      DENVER         CO       1491211888 14384        01/21
01/20/08     TARGET               PETOSKY        MI       158249430 613034        01/21
01/20/08     UNITED AIRLINES      DENVER         CO       1451776003 12954        01/21
01/20/08     WORLD TRAVEL         LOUISVILLE     KY       344908098 779883        01/21
01/21/08     CONOCO OIL CO.       NASHVILLE      TN       430479315 702226        01/22
```

Figure 1-8 Sample WEBi window showing a retrieved customer report

Figure 1-9 shows a sample WEBi window with a retrieved bank statement.



PAGE 1 OF 1

STATEMENT DATE
07/20/08
CREDIT LIMIT
8000.00

CREDIT AVAILABLE	AMOUNT OVER LIMIT
7722	0

ACCOUNT NUMBER
000-000-152

MAKE CHECKS PAYABLE TO: BigBucksCard

DIRECT PAYMENT OR INQUIRIES TO: BigBucksCard Center
CUSTOMER SERVICES
P.O BOX 100/BOULDER, CO 99999
PHONE (123) 123-4567

A & T PIANO CO
79 S MAIN
LODI, NJ 07644

Figure 1-9 Sample WEBi window showing a retrieved bank statement

2

ODWEK Java API classes

In this chapter, we present the classes of the OnDemand Web Enablement Kit (ODWEK) Java APIs and explain how they are related to one another. We describe the essential use of these classes and explain how to set up development environment projects for ODWEK-enabled applications.

We cover the following topics:

- Core API classes and their functional relationship
- Sample console application
- Setting up a Web development environment by using Rational Application Developer

2.1 Core API classes and their functional relationship

All ODWEK Java API classes are in the com.ibm.edms.od Java package. By convention, commonly used ODWEK Java classes (ODFolder, for example) are considered individual APIs. In this section, we introduce the classes at the core of the ODWEK Java APIs.

API Javadoc™ documentation: The documentation for the APIs is shipped as part of the ODWEK installation. The documentation is in the Javadoc format and must be extracted from the <*odwek installation*>/api/ODApiDoc.zip file. After extracting the files, open the index.html file in a browser to view the Javadoc content.

The sections below survey the highlights of selected APIs. Consult the Javadoc documentation for a complete list of available methods.

2.1.1 Core API classes

The following API classes are the most commonly used according to their typical use:

- Server connection classes
- Content Manager OnDemand data model classes
- Search classes
- Document data retrieval classes
- Error handling class

Server connection classes control connections to a Content Manager OnDemand (OnDemand) server. This group includes the following classes:

- ODServer
- ODConfig

Data model classes expose the OnDemand data model and metadata that relates to document storage. This group includes the following classes:

- ODCabinet
- ODFolder
- ODApplication
- ODApplicationGroup
- ODApplicationGroupField

Search classes are used to perform document searches. This group includes the following classes:

- ODFolder
- ODCriteria
- ODNamedQuery
- ODNamedQueryCriteria

Document data retrieval classes convey search results and are used to retrieve OnDemand documents. This group includes the following classes:

- ODHit
- ODHitProperties
- ODNote

The *error handling class* consists of the *ODException class,* which handles Content Manager OnDemand exception errors.

2.1.2 Functional relationship

Figure 2-1 on page 24 illustrates important functional relationships among these core classes. Follow the italicized method calls along the highlighted path to trace the usual connect-search-retrieve workflow.

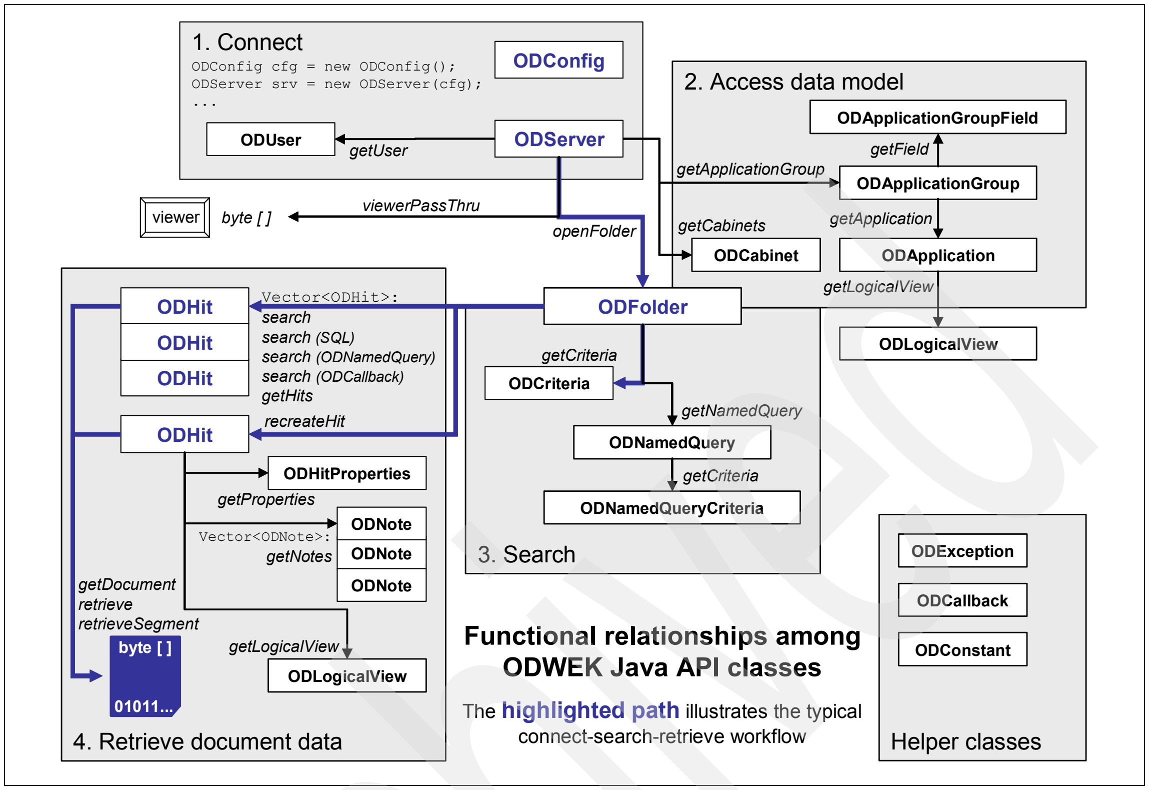


Figure 2-1 API functional relationships

In the next several sections, we take a closer look at some of these classes.

2.1.3 Server connection classes

By using the ODConfig and ODServer classes, you can connect to and disconnect from OnDemand servers.

ODConfig

Use an instance of the ODConfig class to configure a connection to Content Manager OnDemand. Prior to Content Manager OnDemand version 8.4, these settings were specified in the arswww.ini file. In version 8.4, the proper way to specify connection parameters is to use the ODConfig object.

With the parameters of the ODConfig class, you can specify such settings as the conversion mechanism to be used for document data viewing, the ODWEK's temporary directory, and the level of detail for runtime log files.

The ODConfig class has three constructors:

- ODConfig() with no parameters

  Instantiating ODConfig with no parameters initializes the configuration with default values (see Table 2-1).

- ODConfig() with all parameters except the advanced properties

  An ODConfig object can be instantiated with all the parameters listed in Table 2-1 without specifying the optional properties. If there is no requirement for AFP2PDF or AFP2WEB data conversions, then use this constructor.

- ODConfig() with all parameters

  If you want to use AFP2PDF or AFP2WEB document conversions, then pass properties that specify AFP2xxx parameters.

Table 2-1 shows the meaning of the constructor parameters and their default values.

Table 2-1 Meaning of the ODConfig constructor parameters

Data type	Parameter	Default value
String	AFP data conversion type	ODConstant.PLUGIN
String	LINE data conversion type	ODConstant.APPLET
String	META data conversion type	ODConstant.NATIVE[a]
long	Maximum hits	200
String	Applets directory	`"/applets"`
String	Language for output messages	`"ENU"`
String	Temp directory	Value returned by `System.getProperty("java.io.tmpdir")`
String	Absolute path for trace file	Value returned by `System.getProperty("java.io.tmpdir")`
int	Trace level	0
java.util. Properties	Advanced configuration properties	optional

a. ODConstant.NATIVE can be used for those customers who want AFP data in its native format.

The ODConfig object offers access to the configuration settings that it stores. See Table 2-2 for the ODConfig access methods.

Table 2-2 ODConfig access methods

Data type returned	Method and description
String	getAFPViewerType() Get the AFP viewer type.
String	getAppletDirectory() Get the URL path or alias of the applet JAR file.
String	getLanguageForMessages() Get the three character language abbreviation for messages.
String	getLineViewerType() Get the Line viewer type.
long	getMaxNumberOfHitsToDisplay() Get the maximum number of search result hits.
String	getMetaViewerType() Get the META viewer type.
String	getTemporaryWorkingDirectory() Get the temp directory for OnDemand working files.
String	getTraceDirectory() Get the directory in which to place the arswww.trace files.
int	getTraceLevel() Get the trace level, which determines the detail of the arswww.trace content.
void	printConfig() Print the configuration settings.

ODServer

The ODServer class manages a connection to an OnDemand server. No other API class (except ODConfig) can be instantiated until a valid connection has been established through the ODServer object. If a connection becomes invalid, perhaps timed out due to inactivity, the same ODServer instance can be used to log on again.

Connected ODServer instances reserve native system resources. It is important to manage them so as not to cause a resource or memory leak. Depending upon your application, connections can be pooled, stored in an HTTP session, or simply used once and disposed. We discuss various connection management techniques in the following chapters.

Prior to Content Manager OnDemand version 8.4, ODServer instances could be instantiated with a no- argument constructor. With Version 8.4, you must pass an ODConfig object to create new ODServer instances.

To connect to Content Manager OnDemand:

1. Instantiate a new ODConfig object.
2. Instantiate a new ODServer object by using the ODConfig instance.
3. Call the ODServer.intialize() method, which takes a String parameter.

   In the context of a Web application, pass the name of the Java class that implements the viewer pass-through function for the Java Line Data Viewer applet and the AFP plug-in. If your application does not use the applet or the AFP plug-in, then any string value will suffice. The common practice is to pass the name of the class that instantiates the ODServer.
4. Call the ODServer.logon() method.

   The server name, user ID, and password are the minimum prerequisites for logging on.

After the ODServer connection is no longer required, ensure proper disposal of the connection:

1. Call the ODServer.logoff() method to log off the current user ID from the server.
2. Call the ODServer.terminate() method to terminate and dispose the network connection from the application to the server.

Table 2-3 lists popular ODServer methods.

Table 2-3 Popular ODServer methods

Data type returned	Method and description
void	cancel() Cancel the current search or retrieve operation on this connection.
void	changePassword(String newPassword) Change the logged-on user’s password.
java.util.Enumeration (of `ODCabinet` instances)	getCabinets() Get a list of the cabinets that are defined on the connected server.
int	getNumCabinets() Get the number of cabinets that are defined on the connected server.

Data type returned	Method and description
int	getNumFolders() Get the number of folders that are defined on the connected server.
java.util.Enumeration (of `ODFolder` instances)	getFolders() Get the folders that are defined on the connected server.
ODFolder	openFolder(String fldname) Open a given folder.
String	getFolderDescription(String fldname) Get a given folder's description.
void	initialize(String applicationName) Initialize this connection to the OnDemand server.
void	logon() Log on the configured user.
void	logoff() Log off the configured user.
void	terminate() Terminate this connection to the server.
byte[]	viewerPassthru(String queryString) Return a document to a viewer based on the query string from the applet.
boolean	isServerTimedOut() Get the timeout status of this connection.
boolean	keepServerAlive() Reset the connection timeout.
String[]	getServerPrinters() Get a list of printers that are defined on the server.

2.1.4 Content Manager OnDemand data model classes

The ODCabinet, ODFolder, ODApplicationGroup, ODApplication, and ODApplicationGroupField classes comprise a model of the logical organization of data stored in Content Manager OnDemand. Pay special attention to the ODFolder class because it is the interface to the document search feature.

ODCabinet

ODCabinet instances represent Content Manager OnDemand cabinets, which are defined by the administrator to group a particular set of folders for the convenience of users. Obtain the list of cabinets for your server by calling the ODServer.getCabinets() method. Remember that a folder can belong to multiple cabinets. Searches cannot be performed against the ODCabinet class.

Table 2-4 lists the ODCabinet methods.

Table 2-4 ODCabinet methods

Data type returned	Method and description
String	getDescription() Get this cabinet's description.
String[]	getFolderNames() Get folder names for this cabinet. **Note**: First call open() to populate the folder list.
String	getName() Get this cabinet's name.
int	getNumFolders() Get the number of folders for this cabinet. **Note:** Call the open() method first to retrieve the number of folders for this cabinet.
void	open() Open this cabinet.

ODFolder

ODFolder instances represent Content Manager OnDemand folders. To obtain a usable ODFolder instance, call ODServer.openFolder(). Instances of ODFolder expose the server's definition of folders, and the criteria used to search them for documents. Searches are usually performed by setting values into ODCriteria objects obtained from an ODFolder, and then calling the folder's search() method.

Table 2-5 lists principle ODFolder methods.

Table 2-5 ODFolder methods

Data type returned	Method and description
void	open() Open this folder.
void	close() Close this folder.
String[]	getApplGrpNames() Get the application groups for this folder.
java.lang.Object[] (Returned objects are Strings)	getApplNames(String applGroup) Get names of applications in the given application group.
java.util.Enumeration (of ODCriteria instances)	getCriteria() Get the search criteria for this folder.
String	getDescription() Get this folder's description.
java.util.Vector (of ODHit instances)	getHits() Get the ODHit results of the last search performed by this folder.
ODCriteria	getCriteria(String name) Get a particular search criterion.
ODNamedQuery	getNamedQuery(String name) Get a specific ODNamedQuery object.
java.util.Enumeration (of String)	getNamedQueryNames() Get a list of named query names for this folder.
void	printDocuments(java.util.Vector hits,String printer,int copies) Print the documents referenced by the given ODHit objects.
ODHit	recreateHit(String docid) Recreate an ODHit object given a document ID.
java.util.Vector (of ODHit instances)	search() Perform a search. Call after search criteria are set.
java.util.Vector (of ODHit instances)	search(ODNamedQuery namedQ) Perform a search by using the specified named query.

Data type returned	Method and description
java.util.Vector (of ODHit instances)	search(String sqlWhereClause) Perform an SQL search.
long	searchCountHits() Get the total number of hits that will be returned by a search of this folder by using the currently-specified criteria.

ODApplicationGroup

ODApplicationGroup instances correspond to Content Manager OnDemand application groups. Obtain ODApplicationGroup instances by calling the getApplicationGroup() method on an ODServer instance.

Table 2-6 lists selected ODApplicationGroup methods.

Table 2-6 ODApplicationGroup methods

Data type returned	Method and description
ODApplication	getApplication(String name) Get an ODApplication object associated with this application group.
String[]	getApplicationNames() Get the names of all applications that are associated with this application group.
String	getDescription() Get this application group's description.
java.util.Enumeration (of ODApplicationGroupField instances)	getFields() Get the fields that are defined in this application group.
String	getName() Get the name of this application group.

ODApplication

ODApplication instances correspond to Content Manager OnDemand applications. Obtain these instances by calling the getApplication() method on an ODApplicationGroup instance.

Table 2-7 lists the principle ODApplication methods.

Table 2-7 ODApplication methods

Data type returned	Method and description
String	getDescription() Get this application's description.
char (one of the FileTypeXXX constants defined in interface ODConstant; see the Javadoc documentation)	getDocumentType() Get the document type defined for this application.
String	getName() Get the name of this application.

ODApplicationGroupField

ODApplicationGroupField objects represent OnDemand application group fields. Obtain instances by calling the ODApplicationGroup.getFields() or ODApplicationGroup.getField() methods.

Table 2-8 lists the ODApplicationGroupField methods.

Table 2-8 ODApplicationGroupField methods

Data type returned	Method and description
short	getMask() Get the mask for this application group Field.
String	getName() Get this application group Field's name.

2.1.5 Search classes

In conjunction with the ODFolder class, use the ODCriteria, ODNamedQuery, and ODNamedQueryCriteria classes to prepare and execute document searches.

ODCriteria

Administrators define the search criteria, which ODCriteria objects represent, that are relevant to a given folder. Obtain the set of ODCriteria instances that pertain to a folder by calling the ODFolder.getCriteria() method. Then prepare a search by setting the operator and search value or values for each ODCriteria instance. Search values are passed as strings and are converted by the API for comparison with the corresponding field when executing the search. Note that not all operators apply to all data types.

To search an ODFolder object for documents:

1. Get the desired search criteria by using one of the ODFolder.getCriteria() methods.
2. Set the operator for each ODCriteria instance.
3. Set the search value or values for each ODCriteria. If a criterion refers to a data type field of Date or Time, format the search value string according to the format returned by the ODCriteria.getDefaultFmt() method.
4. Set whether the search criteria will be ANDed or ORed by calling the ODFolder.setOrSearchCriteria() method.
5. Call the ODFolder.search() method to perform the search.

Table 2-9 lists the selected ODCriteria methods.

Table 2-9 ODCriteria methods

Data type returned	Method and description
String[]	getApplicationGroupNames() Get the application group names to which this ODCriteria is mapped.
boolean	getAscending() Get the ascending/descending sort search type for this criterion.
String[]	getFixedValues() Get valid search values for this criterion, if configured.
String	getName() Get this criterion's name.
int	getOperator() Get this criterion's current operator.
String[]	getSearchValues() Get this criterion's current search values.

Data type returned	Method and description
char	getType() Get the type for this criterion.
int[]	getValidOperators() Get the valid operators for this criterion.
boolean	isRequired() Determine whether a search value is required for this criterion.
void	setOperator(int op) Set the operator for this criterion.
void	setSearchValue(String val) Set the search value for this criterion.
void	setSearchValues(String val1, String val2) Set the search values for this criterion when the operator is BETWEEN or NOT BETWEEN.
void	setSortOrder(int val) Set the sort order for the field to which this criterion refers.
boolean	isQueryable() Determine whether this criterion can be used to narrow a search.
boolean	isDisplayable() Determine whether this criterion may be displayed to users.

ODNamedQuery

ODNamedQuery objects represent Content Manager OnDemand named queries. Use ODNamedQuery objects in conjunction with ODFolder to retrieve or execute existing named queries and save new named queries.

Table 2-10 lists the commonly used methods.

Table 2-10 ODNamedQuery methods

Data type returned	Method and description
String	getName() Get this named query's name.
int	getNumCriteria() Get the number of criteria that this named query specifies.
Enumeration (of ODNamedQueryCriteria)	getCriteria() Get this named query's criteria.

ODNamedQueryCriteria

ODNamedQueryCriteria objects represent the criteria of an OnDemand named query. Obtain the criteria by calling the ODNamedQuery.getCriteria() method.

Table 2-11 lists the methods.

Table 2-11 ODNamedQueryCriteria methods

Data type returned	Method and description
String	getName() Get the criterion name.
int	getOperator() Get the search operator.
String[]	getSearchValues() Get the search values.

2.1.6 Document data retrieval classes

The ODHit and ODHitProperties classes convey index data from the OnDemand server. ODHit can also be used to retrieve document content from the server. ODNote objects convey notes associated with a document.

ODHit

ODHit objects represent single Content Manager OnDemand document index entries.

The ODFolder.search() method returns a Java vector that contains the hits, if any, that satisfy the ODFolder object's current search criteria. ODHit objects convey criterion and display values for their containing folder's index fields. The display values are suitable for presentation to users. Be sure to adhere to the folder's display order so that the users' view of document index data is consistent.

Each ODHit object stores a document ID (docid) that uniquely identifies the document on the server. Given this document ID, you can obtain the corresponding ODHit object with a call to the ODFolder.recreateHit() method. Note that the document IDs are occasionally altered by server operations. Therefore, it is unwise to store them indefinitely as persistent bookmarks.

Table 2-12 lists the most commonly used ODHit methods.

Table 2-12 Principle ODHit methods

Data type returned	**Method and description**
char (one of the FileTypeXXX constants defined in interface ODConstant; see the Javadoc documentation)	getDocType() Get document type for this hit.
byte[]	getDocument() Get document content for this hit.
String	getFolderName() Get the name of the folder associated with this hit.
String	getMimeType() Get the MIME type for the document referenced by this hit.
Vector	getNotes() Get notes associated with the referenced document.
String[]	getPrinterNames() Get list of server printers specific to this hit.
ODHitProperties	getProperties() Get the OnDemand internal properties for this hit.
void	getResources(String fileName) Get AFP resources for this hit, if applicable.
byte[]	retrieve(String viewer) Retrieve the referenced document.
byte[]	retrieveSegment(int segment) Retrieve document content for a segment of a large object for this ODHit.
int	getNumSegments() Get number of segments for this hit.
String	getDisplayValue(String criteriaName) Get the given display field value.
String	getDocId() Get the persistent document ID. **Attention**: If a document is updated or otherwise modified, this document ID can and will change.

ODHitProperties

The ODHitProperties object represents internal property values of an ODHit object. Obtain the property information by calling the ODHit.getProperties() method.

Table 2-13 lists the commonly used ODHitProperties methods.

Table 2-13 Commonly used ODHitProperties methods

Data type returned	Method and description
String	getApplicationGroupName() Get the name of the referenced document's application group.
String	getApplicationName() Get the name of the referenced document's application.
long	getLength() Get the full length of the referenced document in bytes.
String	getLoadName() Get the load ID; this value is shared by all documents that are loaded in the same arsload batch.

ODNote

ODNote objects convey OnDemand document notes. Obtain the document note information by calling the ODHit.getNotes() method.

Table 2-14 lists selected methods on the ODNote class.

Table 2-14 Selected ODNote methods

Data type returned	Method and description
String	getDateTime() Get the date and time this note was created.
int	getPageNum() Get the page number for this note.
String	getUserId() Get the creator of this note.
boolean	isPublic() Get whether this note is public.
String	getText() Get this note's text.

Data type returned	Method and description
void	setText(String text) Set this note's text.
void	setPublic(boolean isPublic) Set whether this note is public.

2.1.7 Error handling class

The APIs specify only one custom exception class, which is the ODException class.

ODException

The ODException class is for all exceptions thrown by the ODWEK Java APIs. If ODWEK's logging is enabled, the APIs log diagnostic information to disk when an ODException is created. Application programs should place all API calls within try and catch blocks or at least bracket every logical unit of work such as a document search.

When you catch an ODException, ensure that your application does not leak native resources through the ODServer instance. That is, no ODServer can go out of scope for garbage collection before its logoff() and terminate() methods are called.

Table 2-15 shows ODException methods.

Table 2-15 ODException methods

Data type returned	Method and description
int	getErrorId() Get the error ID code.
String	getErrorMsg() Get the error message.

Tip: For information about Content Manager OnDemand error codes, see *IBM Content Manager OnDemand: Messages and Codes*, SC27-1379, which contains detailed descriptions. Although the guide lists the error codes with the prefix ARS as used in the OnDemand system log, the integer maps correctly to the error ID that is obtained from an ODException.

2.2 Sample console application

In this section, we show a simple console application that logs on to an OnDemand server and lists the folders that have been defined. This program can be used as a ping utility to test connectivity to the server.

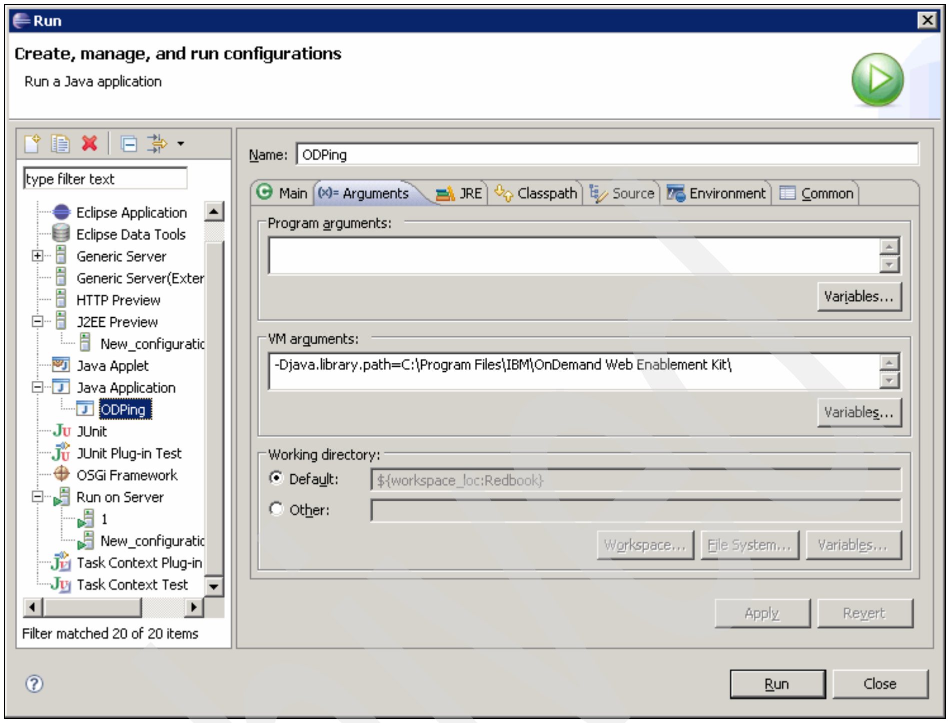
When running this program from the command line, do not forget to specify the java.library.path, so that Java can locate the ARS3WAPI.DLL file. The ICU libraries are loaded via dependencies and must be in the default library search path. Specify the java.library.path parameter by using the -D command line option, as in the following example (on Windows):

```
java -classpath ".;C:\Program Files\IBM\OnDemand Web Enablement
Kit\api\ODApi.jar" ODPing -Djava.library.path="C:\Program
Files\IBM\OnDemand Web Enablement Kit\"
```

Alternatively (though not recommended for production configuration), you can place the native libraries in the default library search path for your platform.

When setting up a development environment for ODWEK console applications such as this example, no special Java project configuration is required. Simply create a plain Java project. You have to reference the ODApi.jar file in your project's classpath or launch configuration and the native library path to the ODWEK shared libraries.

In the Run window of an Eclipse-based integrated development environment (IDE), you can specify the `-D` VM parameter and other launch configuration parameters, as shown in Figure 2-2 on page 40. To access the Run window, from the menu select **Run** → **Open Run Dialog**.



Create, manage, and run configurations

Run a Java application

Name: ODPing

Program arguments:

VM arguments:

```
-Djava.library.path=C:\Program Files\IBM\OnDemand Web Enablement Kit\
```

Working directory:

[✓ Default:] ${workspace_loc:Redbook}

Other:

Figure 2-2   VM arguments in the Run window

Example 2-1 shows the source code for the ODPing console application. After you run this code, you see a message like the following example along with a list of the folders that are defined on your OnDemand server:

```
OnDemand Server mydocserver is alive
```

Example 2-1   ODPing demonstration program

```
import java.util.Enumeration;

import com.ibm.edms.od.ODConfig;
import com.ibm.edms.od.ODConstant;
import com.ibm.edms.od.ODException;
import com.ibm.edms.od.ODFolder;
import com.ibm.edms.od.ODServer;

/**
 * This class demonstrates connecting to an OnDemand
 * server using Version 8.4 of the ODWEK Java API
 * by providing a simple ping console utility.
 */
public class ODPing
{
   public static void main (String[] args) {
      String serverName = "mydocserver"; // Name or IP address
      String userId = "myUsername"; // User ID
      String pwd = "myPassword"; // User password
      int port = 1445; // Default port for OD. Configurable.

      //
      // Configure a new OnDemand server connection with
      // the default configuration for this
      // platform
      //
      ODServer odServer = new ODServer (new ODConfig ());

      try {
         //
         // Set server and log-on credentials
         //
         odServer.setConnectType (ODConstant.CONNECT_TYPE_TCPIP);
         odServer.setServerName (serverName); // Name or IP address
         odServer.setPort (port);
         odServer.setUserId (userId);
         odServer.setPassword (pwd);

         //
         // Initialize the ODServer connection;
         // once initialized, connection MUST be
         // terminated when we're finished.
         // Note use of this class' name, ODPing,
         // as parameter to initialize().
         //
         odServer.initialize ("ODPing");

         //
         // Log on
         //
         odServer.logon ();

         //
         // Report ping status
         //
         System.out.println ("OnDemand server " +
            odServer.getServerName () + " is alive");
         //
         // Perform some useful function
         //
         listFolders (odServer);
      }
      catch (ODException e) {
         //
         // If server returns a "bad credentials"
         // code 2107, then it must be alive
         //
         if (e.getErrorId () == 2107)
         {
            System.out.print ("OnDemand server " +
               odServer.getServerName () + " is alive but: ");
            System.out.println (e.getErrorMsg ());
         }
         else
         {
            System.err.println ("Encountered error: " +
               e.getErrorMsg ());
            System.err.println ("       Error code: " +
               e.getErrorId ());
         }
      }
      catch (Exception e) {
         // Unknown problem
         e.printStackTrace ();
      }
      finally {
         //
         // Ensure user is logged off
         //
         try { odServer.logoff (); }
         catch (Exception e) { /* ignore any problem */ }

         //
         // ALWAYS terminate connections that
         // are no longer needed
         //
         odServer.terminate ();
      }
   }

   public static void listFolders (ODServer odServer)
```

```
      throws Exception
   {
      Enumeration en = odServer.getFolders ();
      while (en.hasMoreElements ()) {
         System.out.println ("Folder: " +
            ((ODFolder) en.nextElement ()).getName ());
      }
   }
}
```

When running the application in your environment, make sure that you update the sample file with your OnDemand server name, user ID, and password (as indicated by use of the bold text in the sample code).

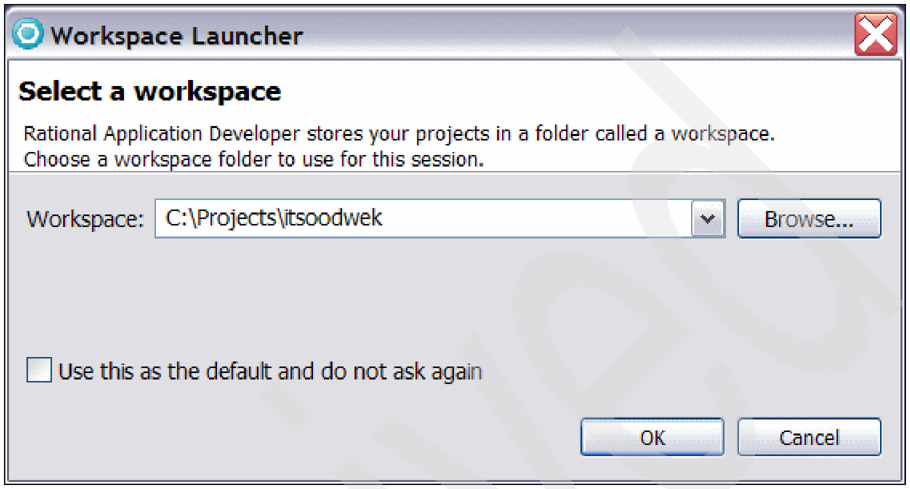
2.3 Setting up a Web development environment by using Rational Application Developer

We use IBM Rational® Application Developer Version 7 in this book to develop code examples. Our development platform consists of the following components:

- Windows XP Professional SP 2.07
- Intel® Pentium® M processor 2 GHz
- 2 GB RAM
- Rational Application Developer V7
- ODWEK V8.4.0.2

In the following example, we step through the creation of a Rational Application Developer Dynamic Web project that you can use to try the code that we present in other chapters of this book. You might find it convenient to create a new workspace to contain your ODWEK application projects.

To create a workspace, in the Rational Application Developer Workspace Launcher window, which is displayed when starting Rational Application Developer (Figure 2-3), click **Browse...** and specify a new, empty workspace directory.


Workspace Launcher

Select a workspace

Rational Application Developer stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Projects\itsoodwek

☐ Use this as the default and do not ask again

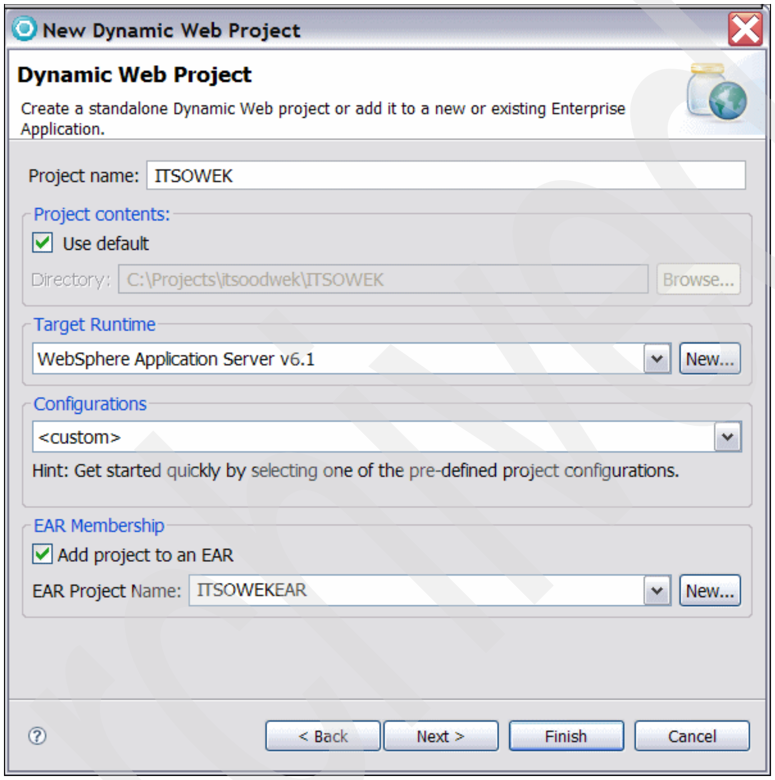
Figure 2-3 Rational Application Developer: Workspace Launcher

After the desired workspace is open, configure a new Web development project.

1. Switch to the Web perspective and create a new Web project:

   a. Select **File** → **New** → **Project** → **Dynamic Web Project**.

   b. In the Dynamic Web Project window (Figure 2-4), enter a project name. In this scenario, we type `ITSOWEK`. Click **Next**.



New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: ITSOWEK

Project contents:

☑ Use default

Directory: C:\Projects\itsoodwek\ITSOWEK

Target Runtime

WebSphere Application Server v6.1

Configurations

<custom>

Hint: Get started quickly by selecting one of the pre-defined project configurations.

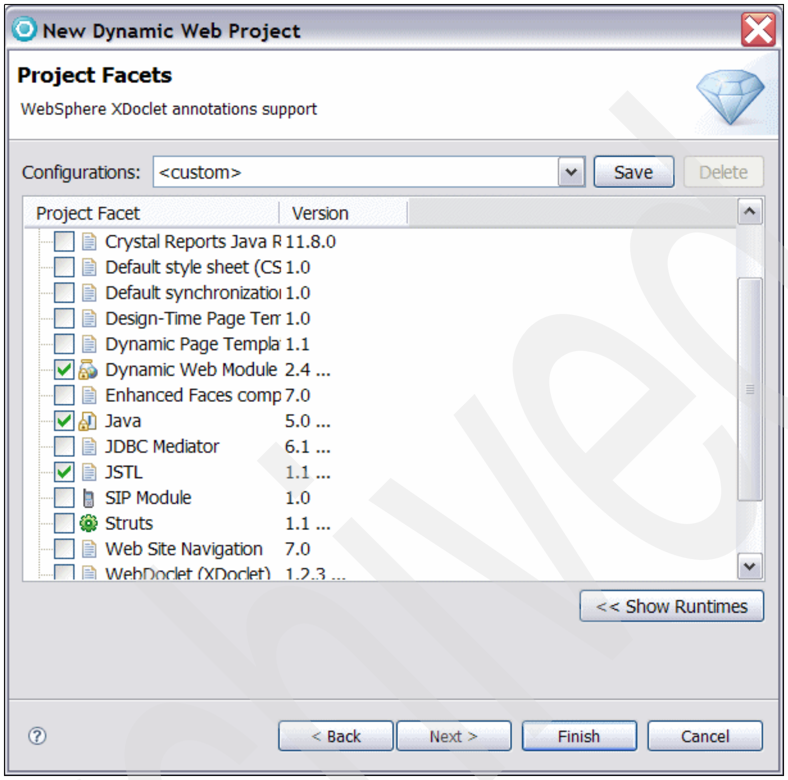
EAR Membership

☑ Add project to an EAR

EAR Project Name: ITSOWEKEAR

Figure 2-4 Creating a new dynamic Web project

c. In the Project Facets window (Figure 2-5), select the **JSTL** check box. Leave the other default facets selected. Click **Next**.


New Dynamic Web Project

Project Facets

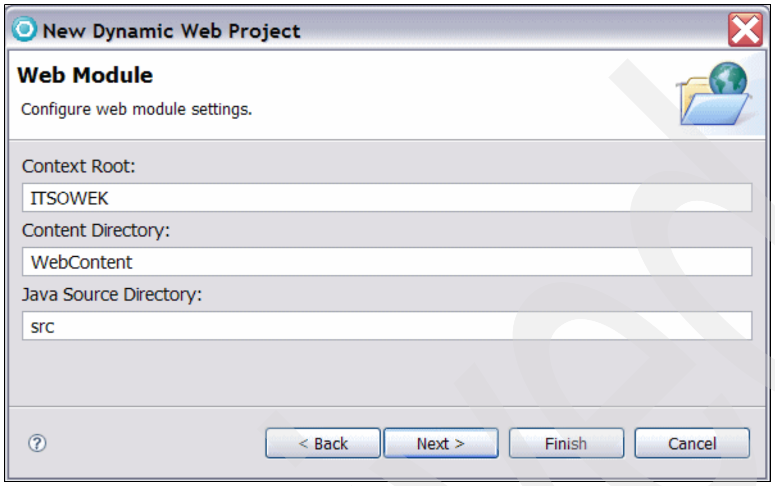
WebSphere XDoclet annotations support

Configurations: <custom>

	Project Facet	Version
☐	Crystal Reports Java R	11.8.0
☐	Default style sheet (CS	1.0
☐	Default synchronizatio	1.0
☐	Design-Time Page Ter	1.0
☐	Dynamic Page Templa	1.1
✓	Dynamic Web Module	2.4 ...
☐	Enhanced Faces comp	7.0
✓	Java	5.0 ...
☐	JDBC Mediator	6.1 ...
✓	JSTL	1.1 ...
☐	SIP Module	1.0
☐	Struts	1.1 ...
☐	Web Site Navigation	7.0
☐	WebDoclet (XDoclet)	1.2.3 ...

Figure 2-5 Selecting additional facets

d. In the Web Module window (Figure 2-6), if necessary, change to the Web context to anything you want. Otherwise, accept the default values and click **Finish**.



Context Root: ITSOWEK
Content Directory: WebContent
Java Source Directory: src

Figure 2-6 Configuring the Web module settings

2. Add the ODWEK Java library as an external reference.

   The project must reference or include the ODWEK Java library classes, which are distributed in the ODApi.jar file. Most often, developers reference the JAR file as an external library rather than importing the class files into their project. With an external reference to the classes, dependent projects automatically reflect updates to ODWEK made by the system administrator. If the library classes are imported into the project, those classes must be re-imported to take advantage of updates.

   To add the ODWEK Java library as an external reference:

   a. Right-click the **ITSOWEK project name** and select **Properties**.

   b. In the Properties window (Figure 2-7 on page 48):

      i. In the type filter text pane on the left, select **Java Build Path**.
      ii. In the right pane, click the **Libraries** tab and click **Add External JARs**.
      iii. Select the **ODApi.jar** file in the <odwek installation root>/api directory.

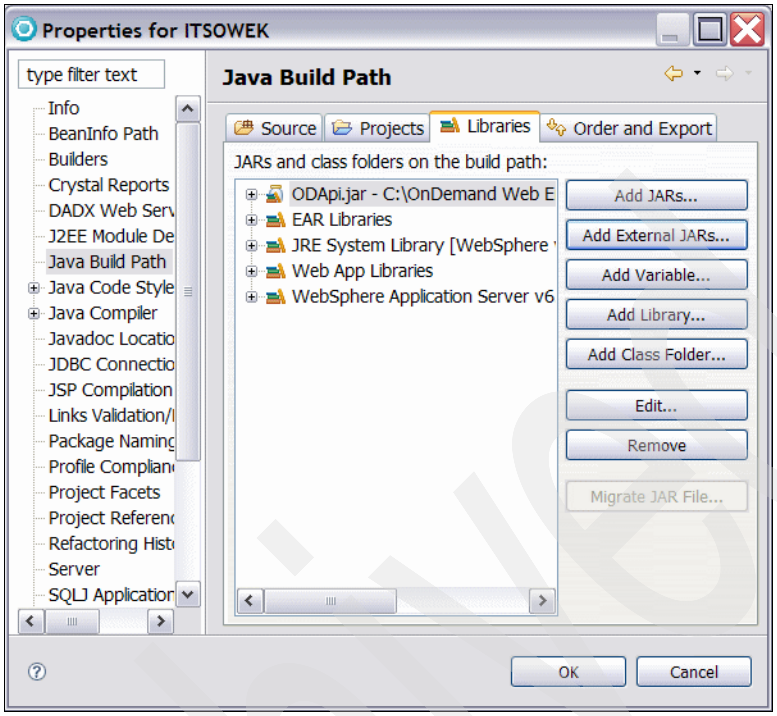
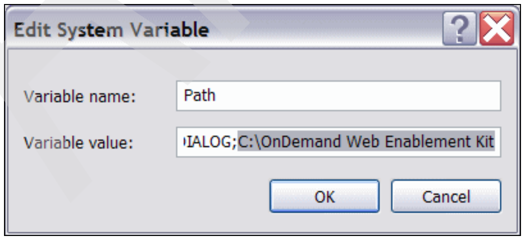


Figure 2-7 Java Build Path

3. In the Edit System Variable window (Figure 2-8), update the path system environment variable to include the <odwek installation root> directory by using the method that is appropriate for your platform.

   The ODWEK library consists of Java classes that rely on native components. One method of referencing the native libraries is to update the system search path to include the ODWEK installation directory.

   Then click **OK**.



Variable name: Path

Variable value: IALOG;C:\OnDemand Web Enablement Kit

Figure 2-8 Updating the path system variable to include the ODWEK base installation directory

4. Add the ITSO ear project (ITSOWEKEAR) to the test server:

   a. Before you start the server, select the **servers** tab, right-click the **WebSphere® Application Server 6.1** server and select **Add and Remove Projects**.

   b. In the Add and Remove Projects window (Figure 2-9), in the Available projects pane, select the **ITSOWEKEAR** project and click **Add** to add the project to the Configured Projects pane. Click **Finish** to save the server configuration.

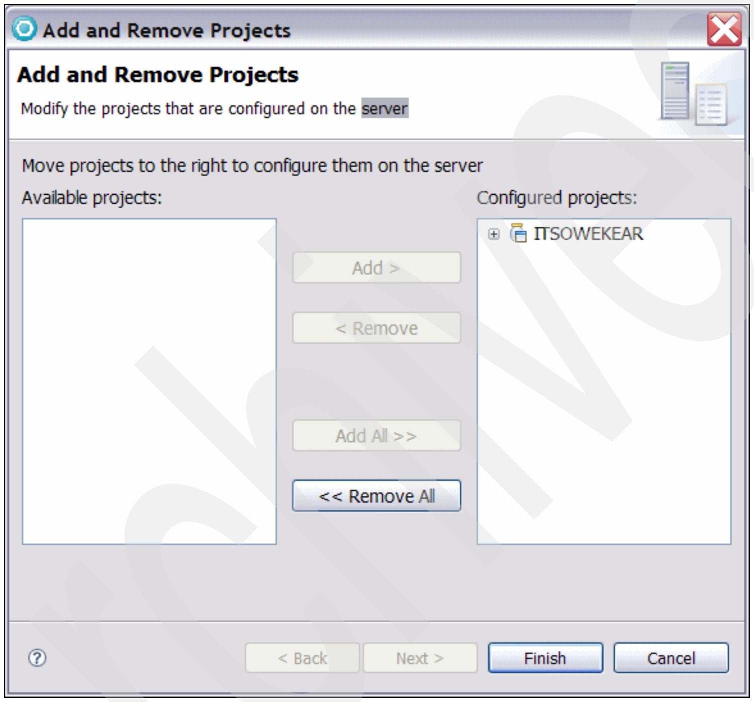


Figure 2-9 Add and Remove Projects window

5. Start the server by selecting **WebSphere Application Server 6.1** and clicking the **Start the server** icon.

   After the server has a status of *started* and a state of *synchronized*, continue with the following step.

6. Add the ODWEK JAR file and native libraries to the application server search path:
   a. Right-click the **server** and select **Run administrative console**.
   b. From left navigation area of the administrative console (Figure 2-10), expand **Environment** and select **Shared Libraries**.
   c. In the right pane, make sure the value of *Scope* reflects `Node=<your node name>` and `Server=server1`. Click the **New** button to name a new shared library.

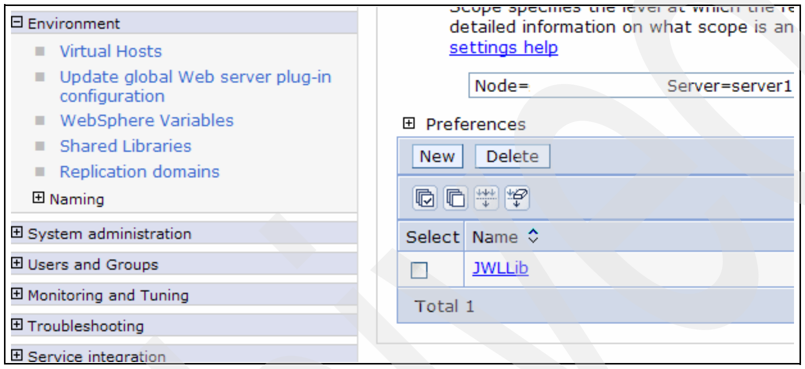
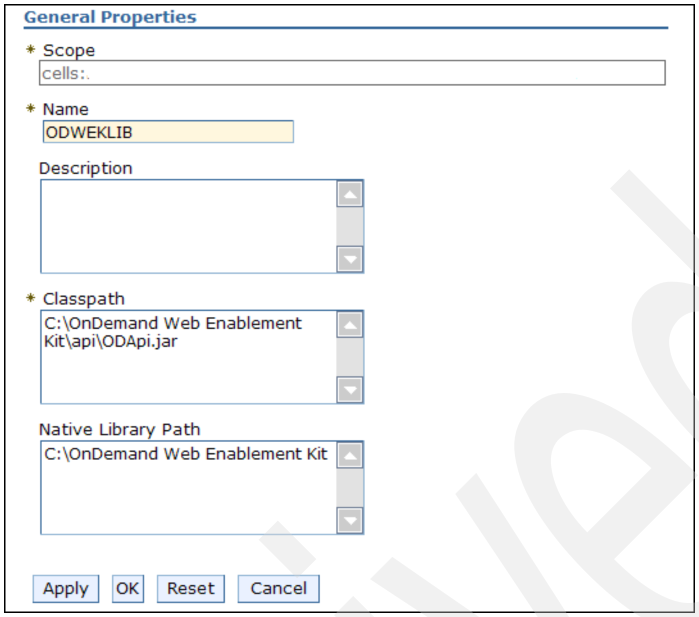


Figure 2-10 Shared libraries

   d. In the next pane (Figure 2-11 on page 51):
      i. For Name, type `ODWEKLIB`.
      ii. For Classpath, type `<odwek installation root>/api/ODApi.jar` for the JAR file.
      iii. For Native Library Path, type `<odwek installation root>`.
      iv. Click **OK**.
      v. Click **Save directly to master configuration**.



General Properties

* Scope

cells:.

* Name

ODWEKLIB

Description

* Classpath

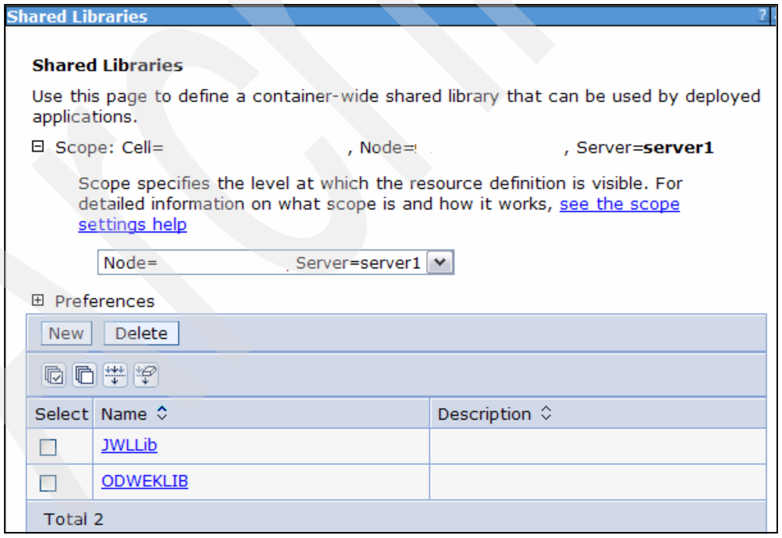
C:\OnDemand Web Enablement Kit\api\ODApi.jar

Native Library Path

C:\OnDemand Web Enablement Kit

Figure 2-11 Adding a new shared library

Figure 2-12 shows the new shared library configuration.



Shared Libraries

Shared Libraries

Use this page to define a container-wide shared library that can be used by deployed applications.

Scope: Cell= , Node= , Server=**server1**

Scope specifies the level at which the resource definition is visible. For detailed information on what scope is and how it works, see the scope settings help

Node= Server=server1

Select	Name	Description
☐	JWLLib	
☐	ODWEKLIB	

Total 2

Figure 2-12 New shared libraries configuration

The project is ready to start adding Web components such as servlets and JavaServer™ Pages.

3

ODWEK Java API examples

In this chapter, we provide examples of the usage of the OnDemand Web Enablement Kit (ODWEK) APIs within a Web application. The application examples are intended to be used for guidance on the different ways that Content Manager OnDemand (OnDemand) data can be accessed.

Important: Use the code as is and at your own discretion. IBM will not provide support for the example code.

This chapter covers the following topics:

- Examples overview
- Making a connection to Content Manager OnDemand
- Obtaining a list of cabinets and folders
- Displaying OnDemand folder information
- Obtaining a list of OnDemand folder search fields
- Displaying an Content Manager OnDemand search results list
- Retrieving and displaying an OnDemand document
- Disconnecting from OnDemand

3.1 Examples overview

For the ODWEK Java API examples in this chapter, servlets are used to illustrate how to enable Content Manager OnDemand with a Web application. Even though you should use more modern frameworks, such as Struts and Faces components, to create Web applications, a basic application that uses servlets and JavaServer Pages (JSP™) files is used throughout these examples.

Rational Application Developer V7 is used as the development workbench. The Java examples use ODWEK V8.4. The Dynamic Web project created in Rational Application Developer in Chapter 2, “ODWEK Java API classes” on page 21, is expanded to add the code examples.

In most cases, the examples that are discussed are depicted as code snippets rather than the entire block of code for the Java class. No attempt is made to create an application that can be used as-is in a production environment in this chapter. The fact that servlets are used in these examples gives you the flexibility to reuse the code snippets as you want within your own framework, such as Struts, with minimal changes.

JSP files are used to render the Web pages. The focus of these discussions is on the actual API calls to Content Manager OnDemand that are done at the servlet level rather than the JSP level. Designing a JSP is not discussed in any detail.

3.1.1 Example files

The Web project has been designed to have Java classes segregated by unique packages. In order to create a package, right-click the **Java Resources: src** folder from the ITSOWEK project and select **new** → **package**. The following packages were created for the purposes of this demonstration:

- The com.ibm.istowek.beans package
- The com.ibm.istowek.servlets package
- The com.ibm.istowek.utils package

Table 3-1 shows a list of the JSPs, servlets, utility files, and Java beans that are used in the example program.

Table 3-1 Files used in the example program

JSPs	Servlets (com.ibm.itsowek.servlets)	Utilities (com.ibm.itsowek.utils)	Beans (com.ibm.itsowek.beans)
logon	ODLogon	ODServerConnection	ODCabinetBean
listfolders	ODInit	ODUtils	ODFolderBean
folderinfo	ODListFolders	ODListener	ODCriteriaBean
searchentry	ODFolderInfo		ODHitBean
searchresults	ODSearchEntry		
hitproperties	ODSearch		
	ODDisplayDocument		
	ODPassthru		
	ODOpenHit		

Java beans

The com.ibm.itsowek.beans package contains Java classes that are used as data Java beans for holding Content Manager OnDemand information as a result of an API call. These bean classes hold information about different OnDemand components such as a folder, cabinet, search criteria, search results, and a document. Table 3-2 lists the beans.

Table 3-2 Beans

Name	Description
ODCabinetBean	Contains getters and setters for information about an OnDemand cabinet.
ODFolderBean	Contains getters and setters for information about an OnDemand folder.
ODCriteriaBean	Contains getters and setters for information about an OnDemand folder field.
ODHitBean	Contains getters and setters for information about an OnDemand hit in the search results list.

Servlets

The com.ibm.itsowek.servlets package contains servlet classes that are used to receive input from an HTTP request, call the appropriate API, and return the OnDemand information back to a JSP. Table 3-3 lists these servlets.

Table 3-3 Servlets

Name	Description
ODLogon	Performs a user logon to Content Manager OnDemand.
ODListFolders	Retrieves all cabinets and folders that a user has permissions to view.
ODFolderInfo	Displays information about a specific folder.
ODSearchEntry	Retrieves the folder field information that a user can use for document searches.
ODSearch	Searches the folder based on the selected search criteria.
ODDisplayDocument	Retrieves the OnDemand document selected from the search results list.
ODOpenHit	Displays information about the selected document from the search results list without retrieving the document.
ODInit	Default servlet that is invoked from the URL to display the logon page.
ODPassthru	Handles a line data applet or AFP viewer callback for document content.

Utility packages

The com.ibm.istowek.utils package contains Java classes that are general purpose utilities that act as helper classes to other classes within the application. Table 3-4 lists the utilities.

Table 3-4 Utilities

Name	Description
ODServerConnection	Performs the API functions to access Content Manager OnDemand.
ODUtils	General purpose methods to perform specialized functions.
ODListener	Implements the HTTPSessionListener.

JSP files

It should be noted that all of the JSPs used in the examples are in the WebContent folder in the ITSOWEK project. Table 3-5 shows the JSPs that are used in these examples.

Table 3-5 JSP mapping

JSP	Mapping
logon.JSP	ODInit servlet
listfolders.JSP	ODListFolders servlet
folderinfo.JSP	ODFolderInfo servlet
searchentry.JSP	ODSearchEntry
searchresults.JSP	ODSearch
hitproperties.JSP	ODOpenHit

Throughout our examples, each servlet has the doGet and doPost methods redirected to a processRequest method to handle either type of HTTP request as shown in Example 3-1.

Example 3-1 Redirecting the doGet() and doPost() methods

```
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
   processRequest(request, response);
}

protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
   processRequest(request, response);
}

protected void processRequest(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
   HttpSession session = request.getSession(true);
```

Each servlet also has a synchronized code block around the API objects to enforce thread safety throughout execution of the application. One way to do this is to synchronize on the HttpSession object as shown in the following example:

```
HttpSession session = request.getSession();
synchronized (session) {
.......code execution.......
} //end synchronized
request.getRequestDispatcher("/listfolders.JSP").forward(request,
response);
```

It is important to note that the HttpSession (session) object is used throughout these examples to store references to other objects in order to persist them between HTTP requests. The ODServer object reference is one example of how to store its reference in the session object. After a successful OnDemand connection is established, then persist the ODServer object in a manner to where it can be retrieved and used among other HTTP requests.

Connection pooling is an extension of how to persist OnDemand connections across multiple HTTP requests within a Web transaction. Use connection pooling to get OnDemand connections, usually for Internet types of applications such as online customer banking. Connection pooling is not used for the examples shown in this chapter, but is explained more in Chapter 6, “Connection pooling and connection handling” on page 111.

3.1.2 Program flow and control

Figure 3-1 shows a summary of the example program flow and control. We explain the details of the program in the following sections.

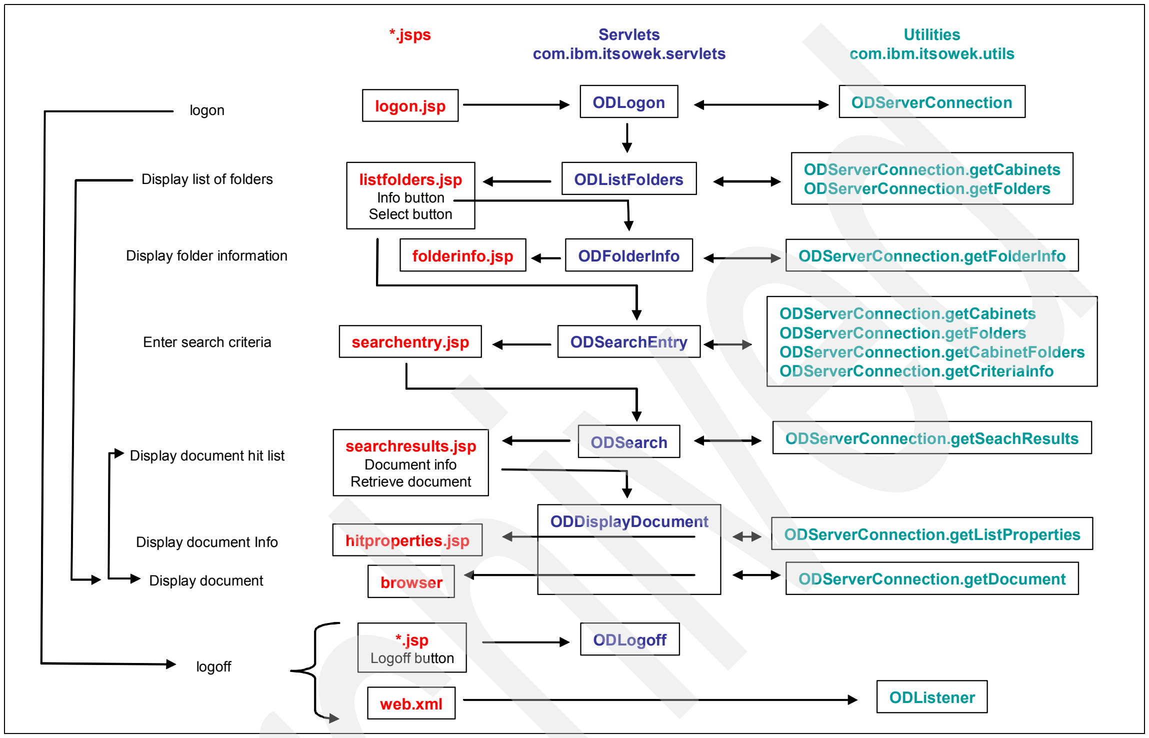


Figure 3-1 Program flow

3.2 Making a connection to Content Manager OnDemand

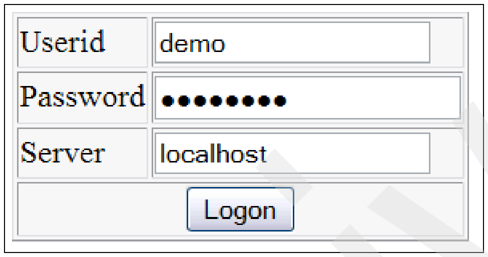
No other API function can be performed until an OnDemand connection is established and remains active. The OnDemand connection is represented by the ODServer object. An active connection is one in which a user ID has been logged on to the target OnDemand server and the logged on connection has not timed out due to inactivity.

You must perform the following steps to make a connection to Content Manager OnDemand:

1. Instantiate an ODConfig object with the correct property values.
2. Instantiate a new ODServer object with the ODConfig object.
3. Initialize the ODServer object.
4. Log on with a valid OnDemand user ID.

The ODConfig object replaces the need to have the arswww.ini configuration settings read in by the application.

Consider the Web page rendered by the logon.jsp file as shown in Figure 3-2.



Userid	demo
Password	●●●●●●●●●
Server	localhost

Figure 3-2 The rendered logon.jsp page

The user supplies the user ID and password and server name. The server name can come from any method you choose. Typically, a list of available OnDemand servers is displayed by name as a selection for the user to choose. These values can come from a properties or XML file that is read in by the application. Then the user clicks Logon.

Then the ODLogon servlet is invoked to get an OnDemand connection. The ODLogon.processRequest method is invoked with request parameters as shown in Example 3-2.

Example 3-2 ODLogon servlet code snippet

```
HttpSession session = request.getSession(true);

synchronized (session) {
   String userid = request.getParameter("userid");
   String password = request.getParameter(“password”);
   String server = request.getParameter(“server”);
   ODServer odServer = null;

   try {
      // get the custom properties
      Properties props = new java.util.Properties();
      java.net.URL url =
ClassLoader.getSystemResource("ODWEK.properties");
      props.load(url.openStream());

      // get the ODServer
      ODServerConnection odServerConnection = new ODServerConnection();
      odServer = odServerConnection.getConnection(server, userid,
password, props,
      request.getContextPath());

   }
```

The servlet receives the request parameters and loads a custom properties file that has application specific directives as shown in Example 3-3.

Example 3-3 ODWEK custom properties

```
# Values used in constructor for ODConfig Object
AfpViewer=plugin
LineViewer=applet
MaxHits=200
MetaViewer=NATIVE
Language=ENU
TempDir=c:\\temp
TraceDir=c:\\temp

# TraceLevels 0=off, 1=minimal, 2=normal, 3=maximum
TraceLevel=3

#for api demo purposes
DefaultServer=localhost
DefaultDisplayFields=

#for internet demo purposes
#DefaultServer=localhost
#DefaultUserid=mamikea
#DefaultPassword=ondemand
#DefaultFolder=Credit Card Statements
#DefaultSearchField=Account
#DefaultSearchDateField=Date
#DefaultDisplayFields=Date,Account Balance

# path to properties for afp2pdf transform
```

```
TransformProperties=C:\\SDP70\\runtimes\\base_v61\\profiles\\AppSrv01\\
properties\\odtransform.properties
```

These directives are used in the ODConfig constructor. We placed these properties in a custom file to dynamically change them without having to change the application code.

The ODServerConnection class controls the API calls. The getConnection method establishes the connection and returns a new ODServer object as shown in Example 3-4.

Example 3-4 ODServerConnection.getConnection() method code snippet

```
ODServer odServer = null;
ODConfig odConfig = null;

// get the custom afp2pdf transform properties
String transformPropsName = myProps.getProperty("TransformProperties");
Properties transformProps = new Properties();
transformProps.load(new FileInputStream(transformPropsName));

// Build the relative URL for the LineDataViewer Applet directory
// This would be something as '/ITSOWEK/applets'
String appletDir = new
StringBuffer().append("/").append(contextRoot).append("/applets").toStr
ing();

// set the configuration values from my custom properties
odConfig = new ODConfig(myProps.getProperty("AfpViewer"), // AfpViewer
      myProps.getProperty("LineViewer"), // LineViewer
      myProps.getProperty("MetaViewer"), // MetaViewer
      Long.parseLong(myProps.getProperty("MaxHits")), // MaxHits
      appletDir, // AppletDir
      myProps.getProperty("Language"), // Language
      myProps.getProperty("TempDir"), // TempDir
      myProps.getProperty("TraceDir"), // TraceDir
      Integer.parseInt(myProps.getProperty("TraceLevel")), // trace
      // level
      transformProps); // properties

odServer = new ODServer(odConfig);
odServer.initialize(new StringBuffer()
   .append("/").append(contextRoot).append("/ODPassthru").toString());
```

```
// logon to the ODServer
odServer.logon(server, userid, password);

// print out the configuration values
// odConfig.printConfig();

return odServer;
```

The TransForm properties parameter on the ODConfig constructor is optional and is only required if an AFP2xxx transformation is performed by ODWEK.

The parameters of the ODServer.initialize() method take on different meanings depending on the requirements of the application. If the ODWEK line data viewer applet or AFP viewer is used for viewing content, specify the name of a servlet that accepts callback calls from these viewers. Otherwise, the parameter can be any string value that is desired.

A counter in ODWEK is incremented for each successful initialize() call and decremented for each terminate() call. During the lib loading process, several internal environments, such as the messaging and trace engines, are also initialized. This is the only time the Language= and TraceDir/TraceLevel settings are read from the ODConfig object. The proper way to end a user's connection to OnDemand is to always call the ODServer.logoff() and ODServer.terminate() methods.

After the servlet receives the new ODServer object, the reference is saved in the session object as shown in the following example:

```
session.setAttribute("odServer", odServer);
```

Finally, if no exceptions are thrown, the servlet forwards the HTTP request and response references to either another servlet or a JSP. In the case of a successful OnDemand connection, the servlet forwards to the ODListFolders servlet as shown in the following example:

```
request.getRequestDispatcher("/ODListFolders").forward(request, 
response);
```

3.3 Obtaining a list of cabinets and folders

An OnDemand folder is used as a search template for stored content. To perform a search, you must obtain and open the ODFolder object. OnDemand user IDs are granted permission to access one or more folders. A list of these folders is available from the ODServer object.

OnDemand cabinets are optional. OnDemand folders can be classified under a higher grouping of cabinets. Folders can belong to one or more cabinets.

The following example shows how to display any existing cabinet and folder information based on the logged on user's permissions. The ODListFolders servlet is used to gather information about cabinets and folders and uses the listfolders.jsp file to render the results on a Web page.

If the ODLogon servlet establishes a successful OnDemand connection, it forwards the request to the ODListFolders servlet, and the processRequest method is invoked as shown in Example 3-5.

Example 3-5 ODListFolders servlet code snippet

```
HttpSession session = request.getSession();

synchronized (session) {
   ODServer odServer = (ODServer) session.getAttribute("odServer");
   try {
      // get cabinets if any exist
      ODServerConnection odServerConnection = new ODServerConnection();
      List listCabinets = odServerConnection.getCabinets(odServer);
      if (listCabinets != null) {
         request.setAttribute("cabinets", listCabinets);
      }
      // get folders
      List listFolders = odServerConnection.getFolders(odServer);
      if (listFolders != null) {
         request.setAttribute("folders", listFolders);
      }
   }
```

The ODListFolders servlet calls the ODServerConnection.getCabinets method to receive a list of ODCabinetBean objects, of which each represents an OnDemand cabinet as shown in Example 3-6.

Example 3-6 ODServerConnection.getCabinets() method code snippet

```
List<ODCabinetBean> listCabinets = new ArrayList<ODCabinetBean>();
for (Enumeration cabinet_enum = odServer.getCabinets();
cabinet_enum.hasMoreElements();) {
   ODCabinetBean odCabinetBean = new ODCabinetBean();
   ODCabinet cabinet = (ODCabinet) cabinet_enum.nextElement();
   cabinet.open();
   odCabinetBean.setName(cabinet.getName());
   odCabinetBean.setDescription(cabinet.getDescription());
odCabinetBean.setFolderNames(Arrays.asList(cabinet.getFolderNames()));
   listCabinets.add(odCabinetBean);
}
return listCabinets;
```

The ODServerConnection.getFolders method is called to receive a list of ODFolderBean objects, of which each represents an OnDemand folder as shown in Example 3-7.

Example 3-7 ODServerConnection.getFolders() method code snippet

```
List<ODFolderBean> listFolders = new ArrayList<ODFolderBean>();
for (Enumeration folder_enum = odServer.getFolders();
folder_enum.hasMoreElements();) {
   ODFolderBean odFolderBean = new ODFolderBean();
   ODFolder folder = (ODFolder) folder_enum.nextElement();
   odFolderBean.setName(folder.getName());
   odFolderBean.setDescription(folder.getDescription());
   odFolderBean.setApplGrpNamesArray(folder.getApplGroupNames());
   listFolders.add(odFolderBean);
}
return listFolders;
```

The ODListFolders servlet saves the references to the list of ODCabinetBeans and ODFolderBeans in the HTTPRequest object and dispatches the listfolders.jsp as shown in the following example:

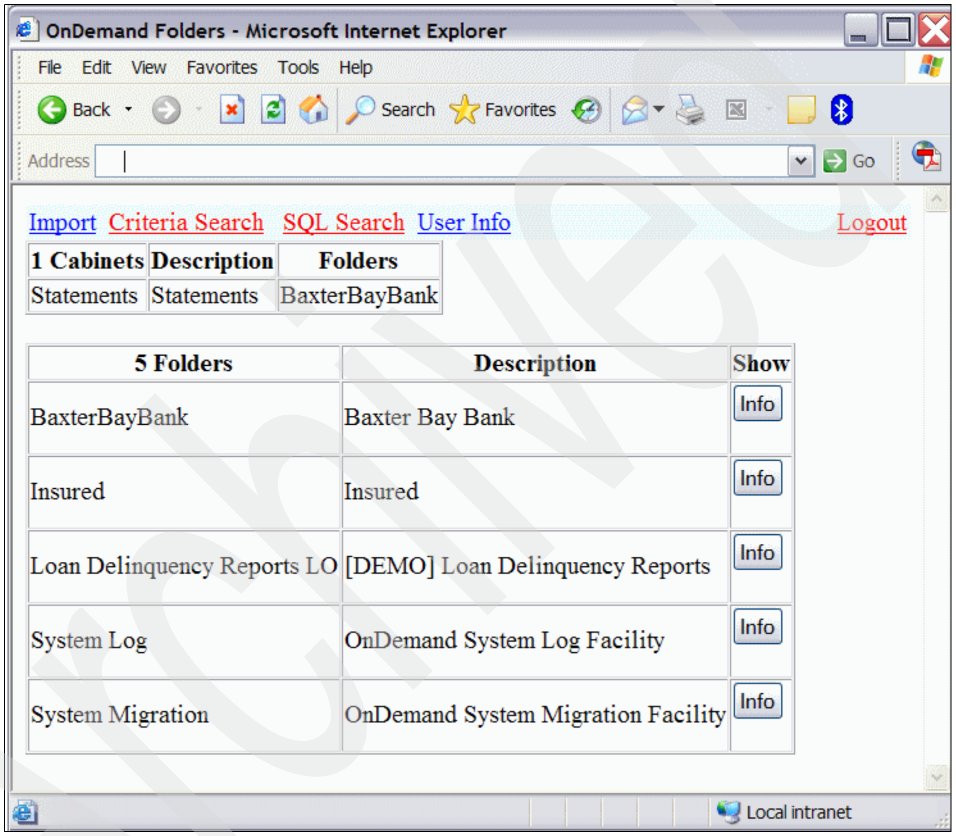
```
request.getRequestDispatcher("/listfolders.JSP").forward(request,
response);
```

By using the JSTL taglib, the JSP can iterate through the list of ODCabinet and ODFolder beans as shown in Example 3-8.

Example 3-8 The listfolders.jsp code snippet

```
<c:if test="${!empty cabinets}">
   <c:forEach var="beanCabinet" items="${cabinets}">
      <tr>
         <td>${beanCabinet.name}</td>
         <td>${beanCabinet.description}</td>
         <td><c:if test="${!empty beanCabinet.folderNames}">
            <c:forEach var="foldername"
items="${beanCabinet.folderNames}">
               ${foldername}
               <br>
            </c:forEach>
         </c:if></td>
      </tr>
   </c:forEach>
</c:if>
```

Figure 3-3 shows the rendering of the JSP file.


OnDemand Folders - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address Go
Local intranet

Import Criteria Search SQL Search User Info Logout

1 Cabinets	Description	Folders
Statements	Statements	BaxterBayBank

5 Folders	Description	Show
BaxterBayBank	Baxter Bay Bank	Info
Insured	Insured	Info
Loan Delinquency Reports LO	[DEMO] Loan Delinquency Reports	Info
System Log	OnDemand System Log Facility	Info
System Migration	OnDemand System Migration Facility	Info

Figure 3-3 The rendered listfolders.jsp page

3.4 Displaying OnDemand folder information

Content Manager OnDemand can return folder information by issuing ODFolder methods to display attribute values such as the number of folder fields and their display order.

In our example, each ODFolder object that is displayed can select the Info button in the listfolders.jsp file. This action invokes the ODFolderInfo servlet processRequest method as shown in Example 3-9.

Example 3-9 ODFolderInfo servlet code snippet

```
HttpSession session = request.getSession();

synchronized (session) {
   ODServer odServer = (ODServer) session.getAttribute("odServer");
   try {
      String folderName = request.getParameter("folder");

      // get folder info
      ODServerConnection odServerConnection = new ODServerConnection();
      ODFolderBean odFolderBean =
odServerConnection.getFolderInfo(odServer, folderName);
      if (odFolderBean != null) {
         request.setAttribute("folderinfo", odFolderBean);
      }
   }
```

The ODFolderInfo servlet calls the ODServerConnection.getFolderInfo() method to get information about the folder from Content Manager OnDemand as shown in Example 3-10.

Example 3-10 ODServerConnection.getFolderInfo() method code snippet

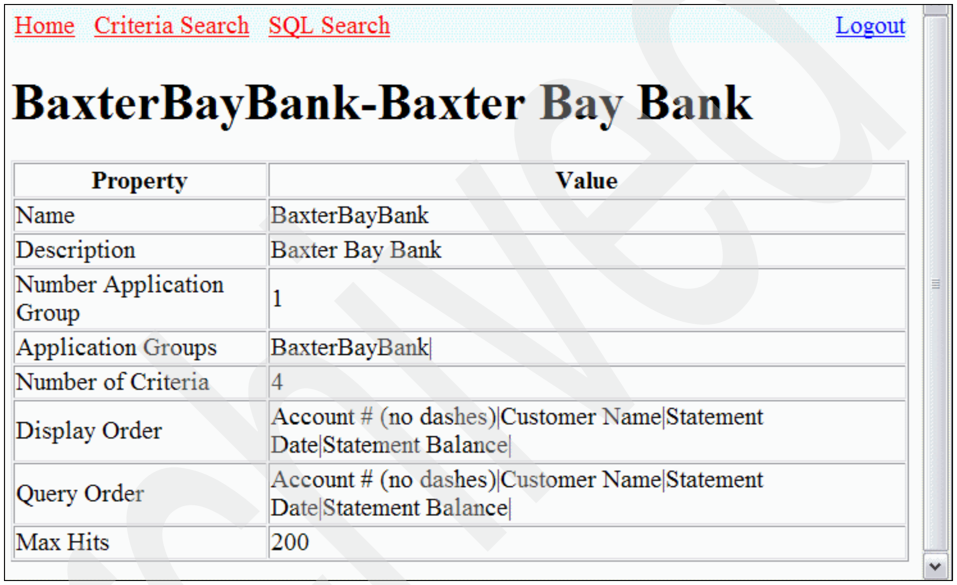
```
ODFolder odFolder = odServer.openFolder(folderName);
ODFolderBean odFolderBean = new ODFolderBean();
if (odFolder != null) {
   odFolderBean.setName(odFolder.getName());
   odFolderBean.setDescription(odFolder.getDescription());
   odFolderBean.setNumApplGroups(odFolder.getNumApplGroups());
   odFolderBean.setNumCriteria(odFolder.getNumCriteria());
   odFolderBean.setMaxHits(odFolder.getMaxHits());

   String[] stringOrder = odFolder.getApplGroupNames();
   odFolderBean.setApplGroupNames(ODUtils.stringArray(stringOrder,
"|"));

   stringOrder = odFolder.getDisplayOrder();
   odFolderBean.setDisplayOrder(ODUtils.stringArray(stringOrder, "|"));

   stringOrder = odFolder.getQueryOrder();
   odFolderBean.setQueryOrder(ODUtils.stringArray(stringOrder, "|"));
}
return odFolderBean;
```

The ODFolderInfo servlet saves the ODFolderBean reference in the request object and dispatches the folderinfo.jsp file. The JSP uses the JSTL taglib to display the information that is held in the bean as shown in Example 3-11.

Example 3-11 The folderinfo.jsp code snippet

```
<c:if test="${!empty folderinfo}">
   <h1>${folderinfo.name}-${folderinfo.description}</h1>
   <TABLE cellpadding="1" cellspacing="1" border="1">
      <TR>
         <TH>Property</TH>
         <th>Value</th>
      </TR>
      <!-- List folder properties  ------->
      <tr>
         <td>Name</td>
         <td>${folderinfo.name}</td>
      </tr>
      <tr>
         <td>Description</td>
         <td>${folderinfo.description}</td>
      </tr>
      <tr>
         <td>Number Application Group</td>
         <td>${folderinfo.numApplGroups}</td>
      </tr>
      <tr>
         <td>Application Groups</td>
         <td>${folderinfo.applGroupNames}</td>
      </tr>
      <tr>
         <td>Number of Criteria</td>
         <td>${folderinfo.numCriteria}</td>
      </tr>
      <tr>
         <td>Display Order</td>
         <td>${folderinfo.displayOrder}</td>
      </tr>
      <tr>
         <td>Query Order</td>
         <td>${folderinfo.queryOrder}</td>
      </tr>
      <tr>
         <td>Max Hits</td>
         <td>${folderinfo.maxHits}</td>
      </tr>
   </TABLE>
</c:if>
```

The folderinfo.jsp file is rendered as shown in Figure 3-4.

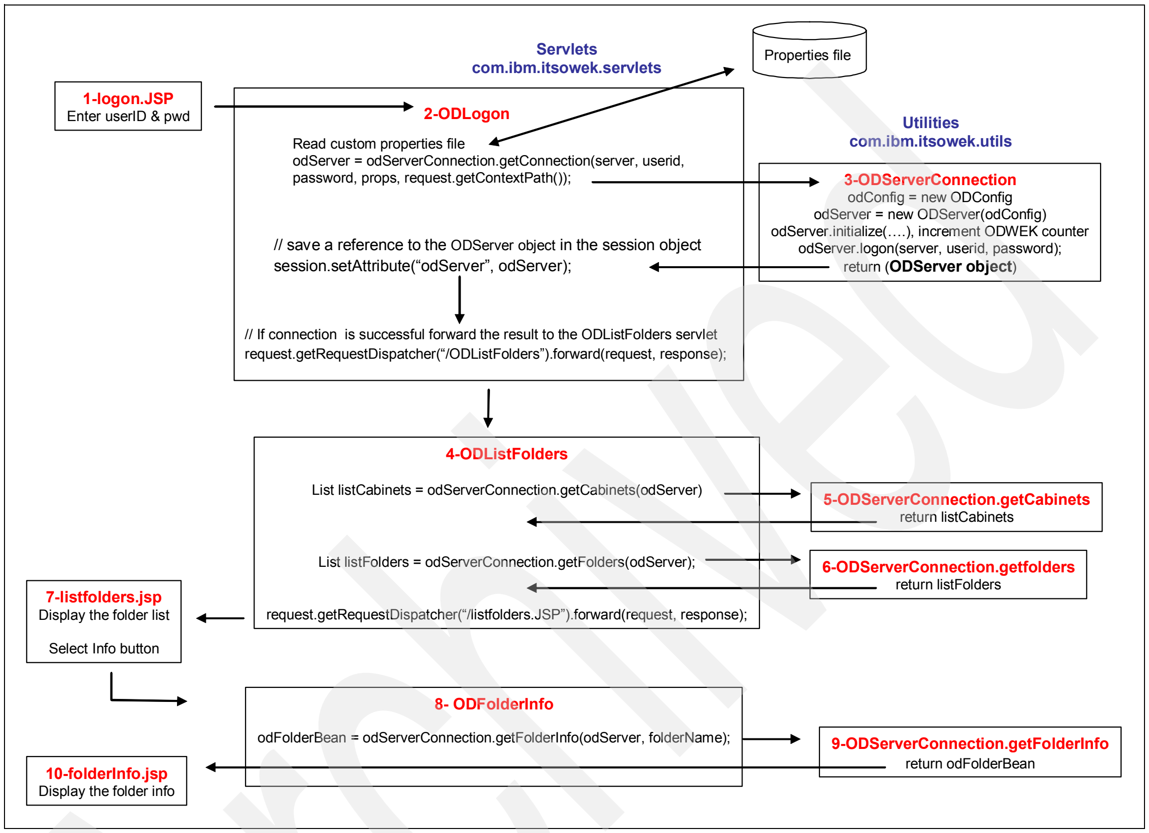


BaxterBayBank-Baxter Bay Bank

Property	Value
Name	BaxterBayBank
Description	Baxter Bay Bank
Number Application Group	1
Application Groups	BaxterBayBank|
Number of Criteria	4
Display Order	Account # (no dashes)|Customer Name|Statement Date|Statement Balance|
Query Order	Account # (no dashes)|Customer Name|Statement Date|Statement Balance|
Max Hits	200

Figure 3-4 The rendered folderinfo.jsp page

Figure 3-5 shows a summary of how the application establishes a connection to the OnDemand server and displays a list of folders that the user is permitted to see.



Servlets
com.ibm.itsowek.servlets
Properties file
1-logon.JSP
Enter userID & pwd
2-ODLogon
Read custom properties file
odServer = odServerConnection.getConnection(server, userid, password, props, request.getContextPath());
// save a reference to the ODServer object in the session object
session.setAttribute("odServer", odServer);
// If connection is successful forward the result to the ODListFolders servlet
request.getRequestDispatcher("/ODListFolders").forward(request, response);
Utilities
com.ibm.itsowek.utils
3-ODServerConnection
odConfig = new ODConfig
odServer = new ODServer(odConfig)
odServer.initialize(….), increment ODWEK counter
odServer.logon(server, userid, password);
return (ODServer object)
4-ODListFolders
List listCabinets = odServerConnection.getCabinets(odServer)
List listFolders = odServerConnection.getFolders(odServer);
request.getRequestDispatcher("/listfolders.JSP").forward(request, response);
5-ODServerConnection.getCabinets
return listCabinets
6-ODServerConnection.getfolders
return listFolders
7-listfolders.jsp
Display the folder list
Select Info button
8- ODFolderInfo
odFolderBean = odServerConnection.getFolderInfo(odServer, folderName);
9-ODServerConnection.getFolderInfo
return odFolderBean
10-folderInfo.jsp
Display the folder info

Figure 3-5 Program flow for connecting to the server and displaying the list of folders

3.5 Obtaining a list of OnDemand folder search fields

To perform an OnDemand folder search, you must set the folder field values with appropriate data via the ODCriteria objects. You can also perform a folder search by setting application group field values by using an SQL search style.

Our example shows how to retrieve the folder fields for display purposes so that a user can complete the appropriate search information. The ODSearchEntry servlet retrieves folder field information to be displayed in the searchentry.jsp file. The user invokes the ODSearchEntry.processRequest() method by clicking the **Search Criteria** link from the listfolders.jsp file.

In this example, the user is given the ability to select a folder from a cabinet to search or to select any folder that the user has permissions to search. For any folder that is chosen, the ODSearchEntry servlet retrieves a list of ODCriteria objects, which represent information about each OnDemand folder field, as shown in Example 3-12.

Example 3-12 ODSearchEntry servlet code snippet

```
HttpSession session = request.getSession();

synchronized (session) {
   ODServer odServer = (ODServer) session.getAttribute("odServer");
   String selectedCabinetName = request.getParameter("Cabinet");
   String selectedFolderName = request.getParameter("Folder");

   try {
      ODServerConnection odServerConnection = new ODServerConnection();
      // get cabinets if any exist
      if (selectedCabinetName != null) {
         List listCabinets = odServerConnection.getCabinets(odServer);
         if (listCabinets != null) {
            request.setAttribute("cabinets", listCabinets);
         }
      }
      // get all alll folders for display or get only cabinet folders
      List listFolders = null;
      if (!selectedCabinetName.equals("List All Folders")) {
         listFolders = odServerConnection.getFolders(odServer);
      }
      else {
         listFolders = odServerConnection.getCabinetFolders(odServer, 
selectedCabinetName);
      }
      if (listFolders != null) {
         request.setAttribute("folders", listFolders);
      }
      // get criteria
      List listCriteria = odServerConnection.getCriteriaInfo(odServer, 
selectedFolderName);
      if (listCriteria != null) {
         request.setAttribute("criterias", listCriteria);
      }
   }
```

The ODSearchEntry servlet calls the ODServerConnection.getCabinets() and getFolders() methods as was performed in the ODListFolder servlet. A call is also made to the ODServerConnection.getCabinetFolders() method to retrieve a list of ODFolder objects that are in the selected cabinet as shown in Example 3-13.

Example 3-13 ODServerConnection.getCabinetFolders() method code snippet

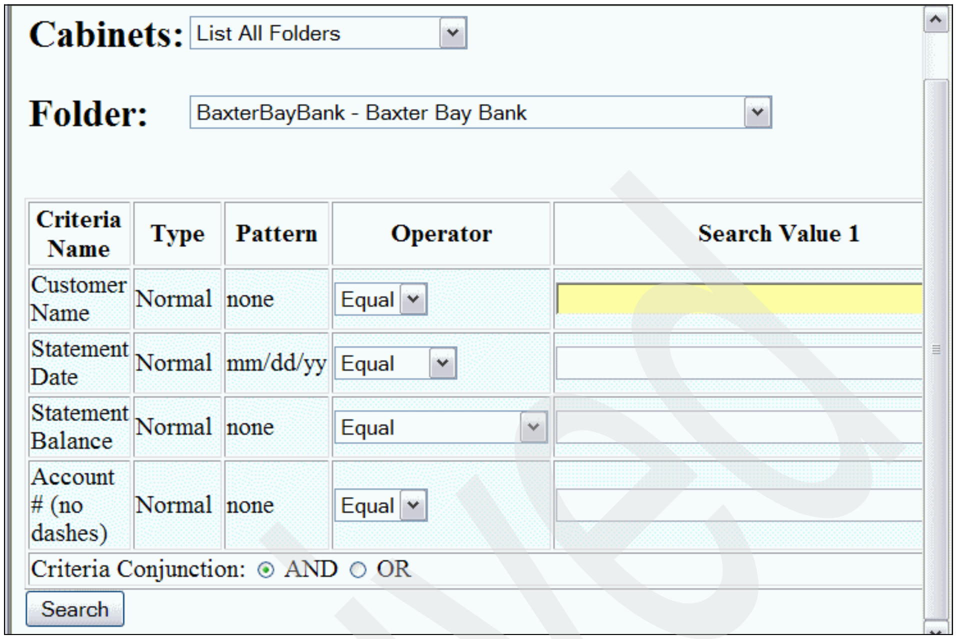
```
List<ODFolderBean> listFolders = new ArrayList<ODFolderBean>();
for (Enumeration cabinet_enum = odServer.getCabinets();
cabinet_enum.hasMoreElements();) {
   ODCabinet cabinet = (ODCabinet) cabinet_enum.nextElement();
   if (cabinet.getName().equals(cabinetName)) {
      cabinet.open();
      String[] folders = cabinet.getFolderNames();
      for (int j = 0; j < cabinet.getNumFolders(); j++) {
         ODFolder folder = odServer.openFolder(folders[j]);
         ODFolderBean odFolderBean = new ODFolderBean();
         odFolderBean.setName(folder.getName());
         odFolderBean.setDescription(folder.getDescription());
         listFolders.add(odFolderBean);
      }
   }
}

return listFolders;
```

The ODSearchEntry servlet calls the ODServerConnection.getCriteriaInfo() method to retrieve a list of folder field objects that are in the OnDemand folder. The getCriteriaInfo() method retrieves the search attributes for each folder field that was defined by the OnDemand administrator. An ODCriteria object is created to represent each folder field that can be queried as shown in Example 3-14.

Example 3-14 ODServerConnection.getCriteriaInfo() method code snippet

```
List<ODCriteriaBean> listCriterias = new ArrayList<ODCriteriaBean>();
// open the slected folder
ODFolder odFolder = odServer.openFolder(folderName);

// get list of odcriteria and iterate
for (Enumeration criteria_enum = odFolder.getCriteria();
criteria_enum.hasMoreElements();) {
   ODCriteriaBean odCriteriaBean = new ODCriteriaBean();
   ODCriteria criteria = (ODCriteria) criteria_enum.nextElement();
   // make sure this is elgible for query display
   if (criteria.isQueryable()) {
      odCriteriaBean.setName(criteria.getName());
      char typeCriteria = criteria.getType();
      odCriteriaBean.setTypeName(ODUtils.getTypeName(typeCriteria));

      // get the operators that have been defined for this field
      int opers[] = criteria.getValidOperators();
      String[] validOperatorNames = new String[opers.length];
      for (int i = 0; i < opers.length; i++) {
         validOperatorNames[i] = ODUtils.operatorName(opers[i]);
      }

odCriteriaBean.setValidOperatorNames(Arrays.asList(validOperatorNames))
;

      // set if this field value must be entered
      odCriteriaBean.setRequired(criteria.isRequired());

      listCriterias.add(odCriteriaBean);
   }
}
return listCriterias;
```

The ODSearchEntry servlet receives a list of ODCriteriaBeans, which represent valid ODCriteria objects that are eligible to be queried. The list is saved as a reference in the request object, and the searchentry.jsp file is displayed as shown in Figure 3-6 on page 74.



Cabinets: List All Folders

Folder: BaxterBayBank - Baxter Bay Bank

Criteria Name	Type	Pattern	Operator	Search Value 1
Customer Name	Normal	none	Equal	
Statement Date	Normal	mm/dd/yy	Equal	
Statement Balance	Normal	none	Equal	
Account # (no dashes)	Normal	none	Equal	

Criteria Conjunction: [✓ AND] OR

Figure 3-6 The rendered searchentry.jsp page

3.6 Displaying an Content Manager OnDemand search results list

After the ODCriteria object values are set, you can perform an OnDemand folder search to return the search results list that is commonly referred to as a *hitlist*.

Our example shows how to receive search information from the searchentry.jsp file and perform a search to obtain a list of objects that represent OnDemand content such as a document. The ODSearch servlet receives search information from the searchentry.jsp file and performs the folder search.

The ODSearch servlet receives the search values and folder names from the searchentry.jsp file. The servlet iterates across each ODCriteria object in the folder and sets each folder field's value or values based on whether the value was specified in the searchentry.jsp file as shown in Example 3-15 on page 75.

Example 3-15 ODSearch servlet code snippet

```
HttpSession session = request.getSession();

synchronized (session) {
   ODServer odServer = (ODServer) session.getAttribute("odServer");
   String selectedFolderName = request.getParameter("Folder");
   String sqlStatement = request.getParameter("SQL_Statement");
   String applGroup = request.getParameter("appl_group");
   String value = null;
   try {
      // open the selected folder
      ODFolder odFolder = odServer.openFolder(selectedFolderName);
      ODFolderBean odFolderBean = new ODFolderBean();
      String[] folderFieldDisplayNames = odFolder.getDisplayOrder();
      odFolderBean.setDisplayOrderArray(folderFieldDisplayNames);
      odFolderBean.setName(selectedFolderName);
      if (sqlStatement != null) {
         odFolder.setApplGroupForSearchWithSQL(applGroup);
      }
      else {
         for (Enumeration criteria_enum = odFolder.getCriteria();
            criteria_enum.hasMoreElements();) {

            // For each search criteria in the folder,
            ODCriteria criteria = (ODCriteria)
criteria_enum.nextElement();
            // The parameter with the criteria name (Name) will have
the
            // value to search for
            value = request.getParameter(criteria.getName());
            // If a value is provided, set the oper and value of the
            // criteria.
            if (value != null && !value.equals("")) {
               // The parameter 'Name_' will have the character
               // representation of the operator integer.
               int oper =
Integer.valueOf(request.getParameter(criteria.getName() + "_"))
                  .intValue();

               criteria.setOperator(oper);
               // The parameter 'Name_2' will have the second value for
               // the BETWEEN/NOTBETWEEN operators
               if (oper == ODConstant.OPBetween || oper ==
ODConstant.OPNotBetween) {
```

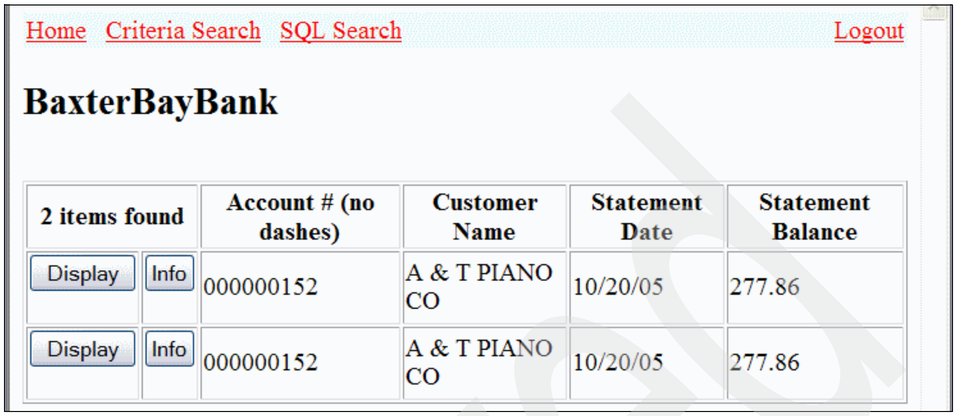
```
                  // For BETWEEN/NOTBETWEEN, set two values in the 
criteria.
                  criteria.setSearchValues(value, 
request.getParameter(criteria.getName() + 
                     "_2"));

               }
               else {
                  // Just set the one value for the search
                  criteria.setSearchValue(value);
               }
            }
         }

         if (request.getParameter("andOr").equals("OR")) {
            odFolder.setOrSearchCriteria(true);
         }
         else {
            odFolder.setOrSearchCriteria(false);
         }
      }
      // Return the search result vector after all criteria is set.
      ODServerConnection odServerConnection = new ODServerConnection();
      List listHits = odServerConnection.getSearchResults(odServer, 
odFolder, sqlStatement,
         odFolderBean.getDisplayOrderArray());

      if (listHits != null) {
         request.setAttribute("hits", listHits);
      }
      if (odFolder != null) {
         session.setAttribute("folderbean", odFolderBean);
      }
   }
```

When the folder field (ODCriteria) values are set, the ODSearch servlet calls the ODServerConnection.getSearchResults() method to perform the OnDemand search. This method generates a list of ODHit objects that represent the OnDemand document information that is returned from a successful search as shown in Example 3-16.

Example 3-16 ODServerConnection.getSearchResults() method code snippet

```
List<ODHitBean> listHits = new ArrayList<ODHitBean>();
// get list of odcriteria and iterate
Vector searchResults = null;
if (sqlStatement != null) {
   odFolder.search(sqlStatement);
}
else {
   searchResults = odFolder.search();
}

if (searchResults != null && searchResults.size() > 0) {
   Iterator iter = searchResults.iterator();
   while (iter.hasNext()) {
      ODHitBean odHitBean = new ODHitBean();
      ODHit odHit = (ODHit) iter.next();
      odHitBean.setDocId(odHit.getDocId());

      // for each folder field to display, get the field value for 
display
      String[] displayVals = new 
String[folderFieldDisplayNames.length];
      for (int i = 0; i < folderFieldDisplayNames.length; i++) {
         displayVals[i] = 
odHit.getDisplayValue(folderFieldDisplayNames[i]);
      }
      odHitBean.setDisplayCriteria(displayVals);
      listHits.add(odHitBean);
   }
}
return listHits;
```

The ODSearch servlet forwards the request to the searchresults.jsp file (Figure 3-7).

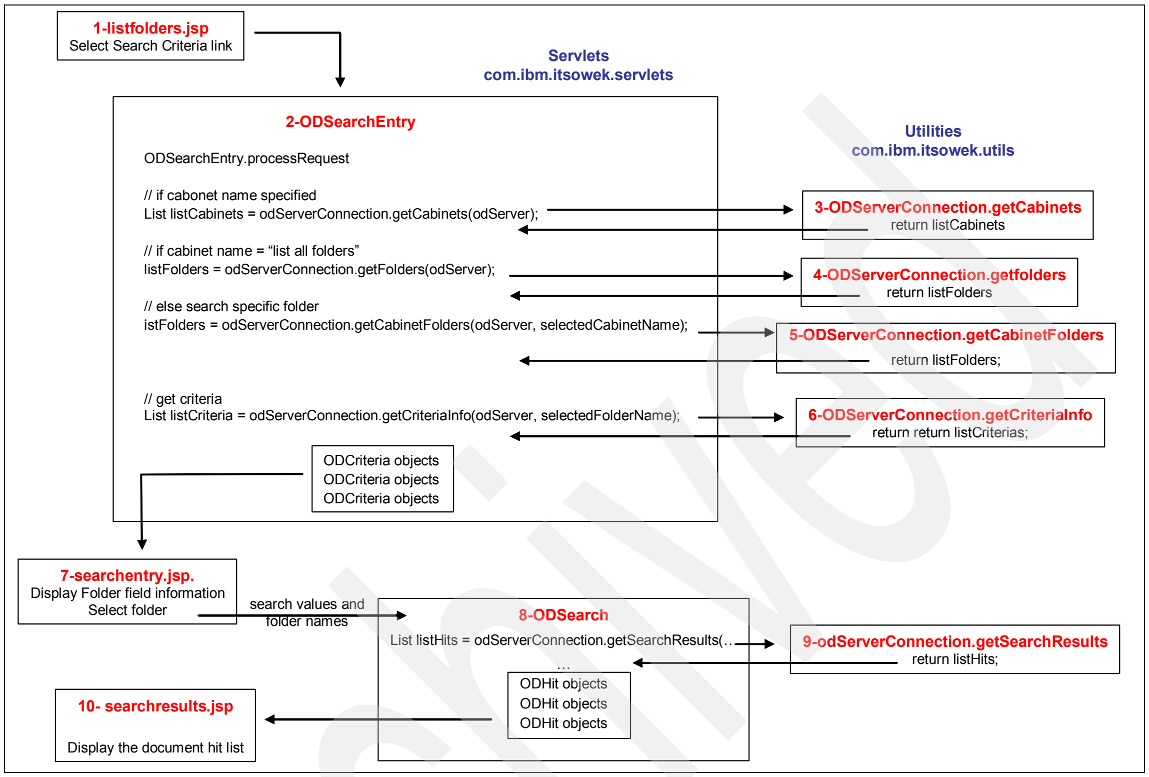


BaxterBayBank

2 items found		Account # (no dashes)	Customer Name	Statement Date	Statement Balance
Display	Info	000000152	A & T PIANO CO	10/20/05	277.86
Display	Info	000000152	A & T PIANO CO	10/20/05	277.86

Figure 3-7 The rendered searchresults.jsp page

Figure 3-8 shows a summary of how the application displays folder search fields and lists the search results.



1-listfolders.jsp
Select Search Criteria link
Servlets
com.ibm.itsowek.servlets
2-ODSearchEntry
ODSearchEntry.processRequest
// if cabonet name specified
List listCabinets = odServerConnection.getCabinets(odServer);
// if cabinet name = "list all folders"
listFolders = odServerConnection.getFolders(odServer);
// else search specific folder
istFolders = odServerConnection.getCabinetFolders(odServer, selectedCabinetName);
// get criteria
List listCriteria = odServerConnection.getCriteriaInfo(odServer, selectedFolderName);
ODCriteria objects
ODCriteria objects
ODCriteria objects
Utilities
com.ibm.itsowek.utils
3-ODServerConnection.getCabinets
return listCabinets
4-ODServerConnection.getfolders
return listFolders
5-ODServerConnection.getCabinetFolders
return listFolders;
6-ODServerConnection.getCriteriaInfo
return return listCriterias;
7-searchentry.jsp.
Display Folder field information
Select folder
search values and
folder names
8-ODSearch
List listHits = odServerConnection.getSearchResults(…
…
ODHit objects
ODHit objects
ODHit objects
9-odServerConnection.getSearchResults
return listHits;
10- searchresults.jsp
Display the document hit list

Figure 3-8 Program flow for displaying search fields and listing the search results

3.7 Retrieving and displaying an OnDemand document

Several methods can be invoked to retrieve and display an OnDemand document. In the example in this section, the document is displayed in its native data format, such as Advanced Function Presentation (AFP). The discussion in Chapter 10, "Applets, plug-ins, and transforms" on page 197, shows how to invoke a transform of an AFP document. Example 3-17 on page 80 shows how to retrieve information about a document by querying its properties. The document properties can be retrieved without retrieving the document content.

Example 3-17 ODDisplayDocument servlet code snippet

```
HttpSession session = request.getSession();
ServletOutputStream out = response.getOutputStream();

synchronized (session) {
   try {
      ODServer odServer = (ODServer) session.getAttribute("odServer");
      String folderName = request.getParameter("folder");
      String docId = request.getParameter("docid");

      ODServerConnection odServerConnection = new ODServerConnection();
      ODHitBean odHitBean = odServerConnection.getDocument(odServer,
folderName, docId,
         true, false);


      response.setContentType(odHitBean.getMimeType());

      // write the byte array out to the browser
      out.write(odHitBean.getOdDocument());
   }
```

Each row in the searchresults.jsp file represents an OnDemand document by using the ODHit object. Each ODHit object has a document ID that is used by Content Manager OnDemand to retrieve the content. Our example shows how to recreate the ODHit object from the document ID that is sent as a parameter from the selected document to view from the searchresults.jsp file.

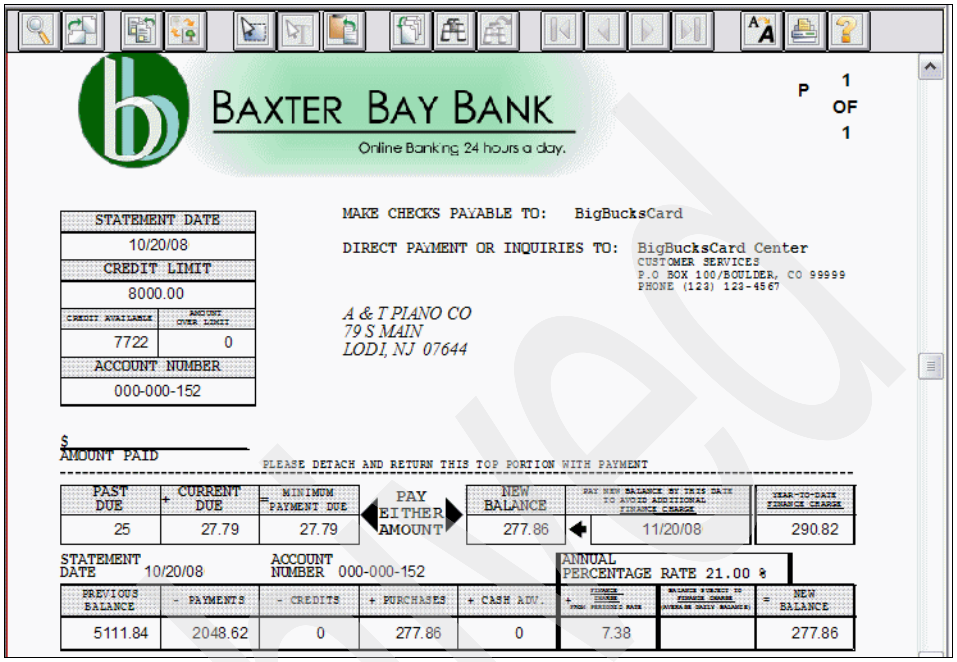
Displaying the document ID on the search results page, either as a visible or hidden html field, can be a security issue for some applications. Another technique to circumvent this issue is to store the ODHit object search results as a list in the HTTPServletSession object without displaying the document ID in the Web page. When the user selects a document from the results list, the ODDisplayDocument matches the relative position of the selected document to the stored ODHit object and uses the relative position to retrieve the document.

The servlet calls the retrieve() method to retrieve the content of the OnDemand document as a byte array. This method is also used to retrieve document information as shown in Example 3-18.

Example 3-18 Retrieve() method code snippet

```
ODHitBean odHitBean = new ODHitBean();
ODFolder odFolder = odServer.openFolder(folderName);

// recreate the ODHit object from the docId
ODHit odHit = odFolder.recreateHit(docId);
odHitBean.setDocId(docId);

// retrieve the content
if (retrieveContent) {
   // retrieve the content
   char fileType = odHit.getDocType();
   String viewerType = "";
   if (transformDoc) {
      viewerType = ODUtils.getTransformDocType(fileType);
   }
   odHitBean.setOdDocument(odHit.retrieve(viewerType));
}
else {
   // retrieve document info only
   getHitProps(odFolder, odHit, odHitBean);
}

// get the mime type
odHitBean.setMimeType(odHit.getViewMimeType());
odFolder.close();

return odHitBean;
```

The result is a display of an OnDemand AFP document that is being rendered by the AFP browser plug-in shown in Figure 3-9.



BAXTER BAY BANK

Online Banking 24 hours a day.

P 1 OF 1

STATEMENT DATE
10/20/08
CREDIT LIMIT
8000.00

CREDIT AVAILABLE	AMOUNT OVER LIMIT
7722	0
ACCOUNT NUMBER	
000-000-152	

MAKE CHECKS PAYABLE TO: BigBucksCard

DIRECT PAYMENT OR INQUIRIES TO: BigBucksCard Center
CUSTOMER SERVICES
P.O BOX 100/BOULDER, CO 99999
PHONE (123) 123-4567

A & T PIANO CO
79 S MAIN
LODI, NJ 07644

$ ______________
AMOUNT PAID

PLEASE DETACH AND RETURN THIS TOP PORTION WITH PAYMENT

PAST DUE	+ CURRENT DUE	= MINIMUM PAYMENT DUE	PAY EITHER AMOUNT	NEW BALANCE	PAY NEW BALANCE BY THIS DATE TO AVOID ADDITIONAL FINANCE CHARGE	YEAR-TO-DATE FINANCE CHARGE
25	27.79	27.79		277.86	11/20/08	290.82

STATEMENT DATE 10/20/08 ACCOUNT NUMBER 000-000-152 ANNUAL PERCENTAGE RATE 21.00 %

PREVIOUS BALANCE	- PAYMENTS	- CREDITS	+ PURCHASES	+ CASH ADV.	+ FINANCE CHARGE FROM PERIODIC RATE	BALANCE SUBJECT TO FINANCE CHARGE (AVERAGE DAILY BALANCE)	= NEW BALANCE
5111.84	2048.62	0	277.86	0	7.38		277.86

Figure 3-9 The rendered AFP document

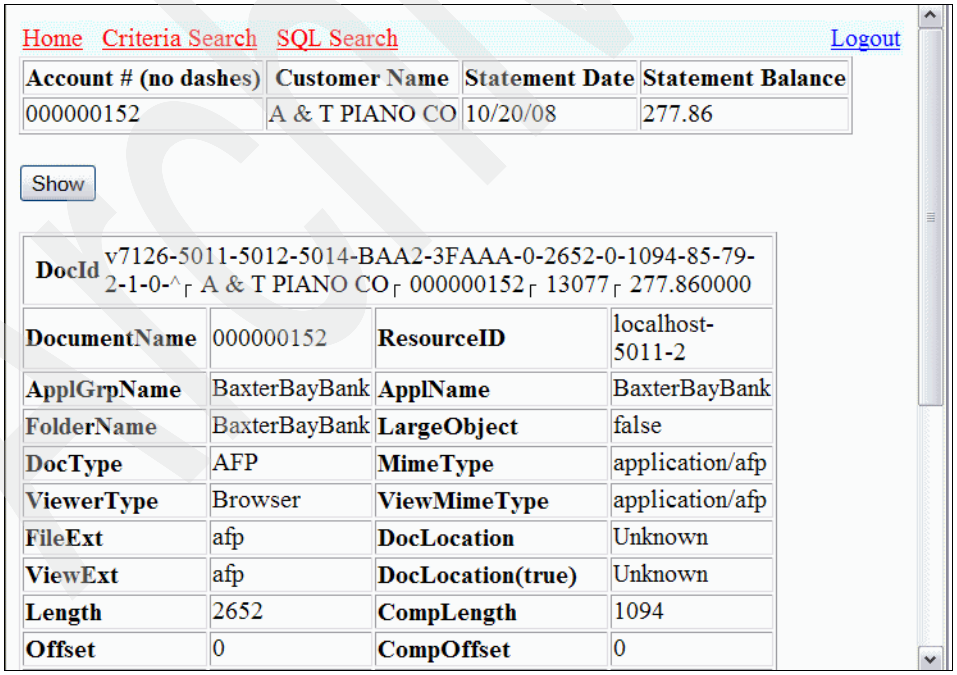
An application can access information about a document that the application might need to know prior to retrieving a document. One such example is interrogating the document length for each ODHit in the search results list. The application might need to know when a document is larger than an acceptable boundary. Based on the information, the application can deny the user the ability to retrieve the document. Then the application can stage the retrieval request during a non-peak time of day and e-mail the document to the user at that time.

When the Info button is selected, the ODDisplayDocument servlet is invoked to retrieve information about the document without retrieving the document content from the OnDemand Object Server. The ODDisplayDocument servlet uses both the ODHit and ODHitProperties APIs to retrieve and consolidate information about the selected document as shown in Example 3-19 on page 83.

Example 3-19   ODServerConnection.getHitProps() method code snippet

```
odHitBean.setResourceID(odHit.getResourceID());
odHitBean.setViewerType(ODUtils.getViewerTypeString(odHit.getViewerType
()));
odHitBean.setViewExt(odHit.getViewExt());
ODHitProperties hitProps = odHit.getProperties();
odHitBean.setLength(String.valueOf(hitProps.getLength()));
odHitBean.setCompLength(String.valueOf(hitProps.getCompLength()));
odHitBean.setCompOffset(String.valueOf(hitProps.getCompOffset()));
odHitBean.setOffset(String.valueOf(hitProps.getOffset()));
odHitBean.setLoadName(hitProps.getLoadName());
odHitBean.setStartDate(hitProps.getStartDate());
odHitBean.setEndDate(hitProps.getEndDate());
odHitBean.setTableName(hitProps.getTableName());

return odHitBean;
odHitBean.setViewMimeType(odHit.getViewMimeType());
```

When the ODDisplayDocument receives an ODHitBean object back, the request is forwarded to the hitproperties.jsp file as shown in Figure 3-10.



Account # (no dashes)	Customer Name	Statement Date	Statement Balance
000000152	A & T PIANO CO	10/20/08	277.86

DocId	v7126-5011-5012-5014-BAA2-3FAAA-0-2652-0-1094-85-79-2-1-0-^┌ A & T PIANO CO┌ 000000152┌ 13077┌ 277.860000		
DocumentName	000000152	**ResourceID**	localhost-5011-2
ApplGrpName	BaxterBayBank	**ApplName**	BaxterBayBank
FolderName	BaxterBayBank	**LargeObject**	false
DocType	AFP	**MimeType**	application/afp
ViewerType	Browser	**ViewMimeType**	application/afp
FileExt	afp	**DocLocation**	Unknown
ViewExt	afp	**DocLocation(true)**	Unknown
Length	2652	**CompLength**	1094
Offset	0	**CompOffset**	0

Figure 3-10   The rendered hitproperties.jsp page

3.7.1 Viewing line data documents with the applet

For line data documents that need to be viewed in their original format, the line data applet is an option to use as the viewer. To use the applet, copy the following directories to your Rational Application Developer project WebContent directory:

- <odwek installation root>/applets
- <odwek installation root>/images

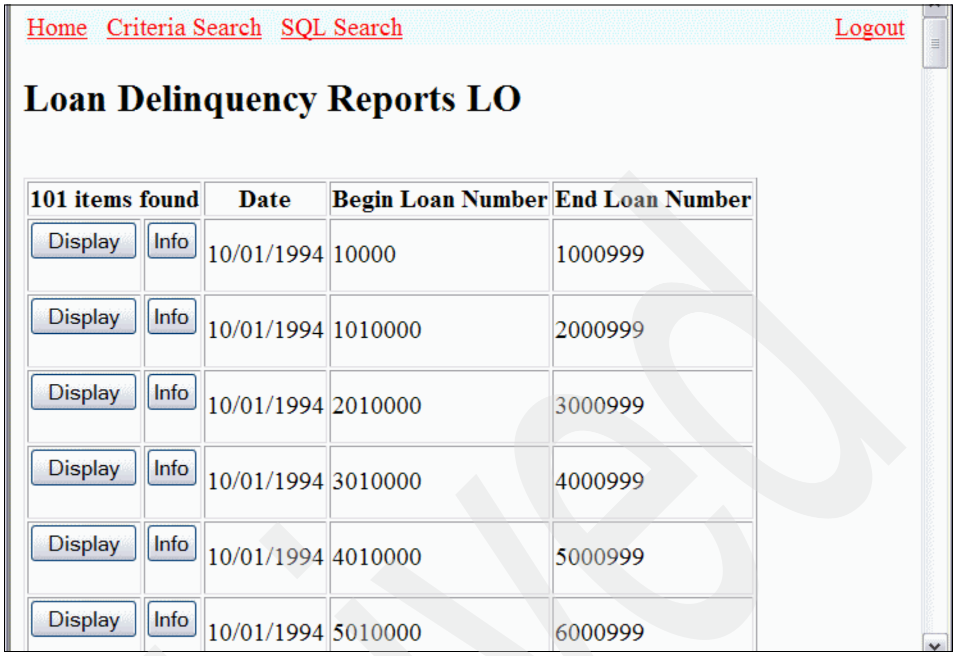
When the applet is invoked by the browser for the first time, it prompts the user with a certificate for approval to run locally on the user's machine. When the applet is loaded, it makes callbacks to the Web application to retrieve the document content and format it locally. The applet makes a callback to a servlet that is identified when the ODServer object was initialized. We initialized the ODServer object as shown in the following example:

```
odServer.initialize(new StringBuffer()
   .append("/").append(contextRoot).append("/ODPassthru").toString());
```

By using our example, the callback is made to /ITSOWEK/ODPassthru.

Note: The viewerPassthru() mechanism is also used by the AFP plug-in.

Line data can be searched and retrieved from the Loan Delinquency Reports LO folder and searched by the default criteria set by the administrator. Figure 3-11 on page 85 shows the search results page.



Loan Delinquency Reports LO

101 items found		Date	Begin Loan Number	End Loan Number
Display	Info	10/01/1994	10000	1000999
Display	Info	10/01/1994	1010000	2000999
Display	Info	10/01/1994	2010000	3000999
Display	Info	10/01/1994	3010000	4000999
Display	Info	10/01/1994	4010000	5000999
Display	Info	10/01/1994	5010000	6000999

Figure 3-11 The rendered searcresults.jsp page

When a user wants to display a line document, the applet is retrieved, and the call back is made from within the HTML as shown in Example 3-20.

Example 3-20 Line data viewer applet HTML

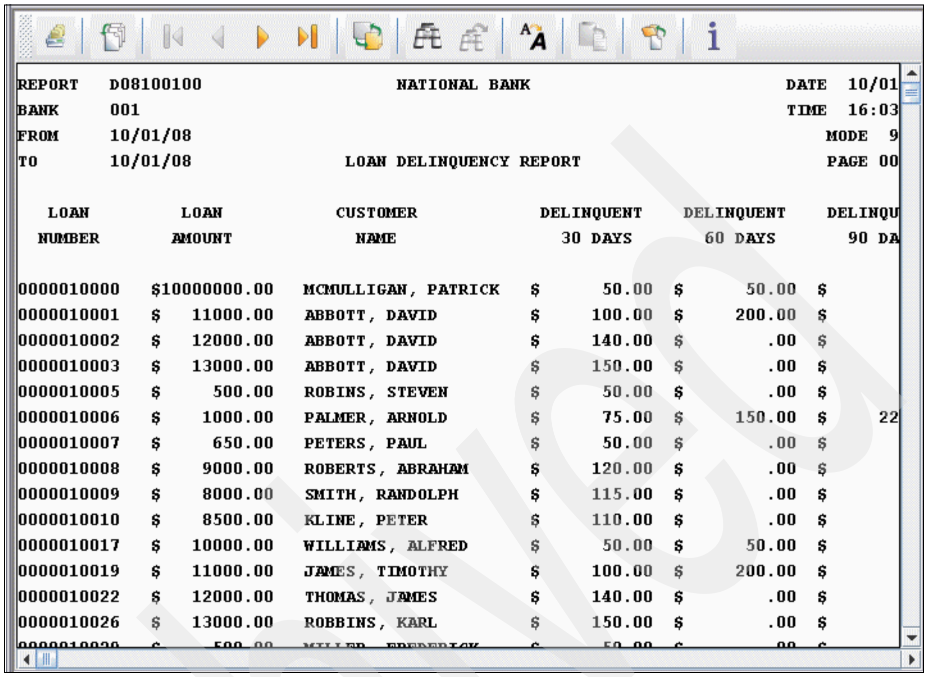
```
<html>
<head>
<script src="/ITSOWEK/applets/IEFix.js"></script>
<title>ODLineDataViewer Applet </title>
</head>
<body onload="getHistoryCnt();"
topmargin="0" leftmargin="0" marginwidth="0" marginheight="0">
      <div id="APPLET_DIV_ID">
      </div>
   <script>
   CreateControl("APPLET_DIV_ID",
                 "http://java.sun.com/getjava/",
                 "com.ibm.edms.od.ODLineDataViewer",
                 "/ITSOWEK/applets/",
                 "1.4.1",
                 "ODLineDataViewer2.jar",
                 "/applets/images",
                 "",
                 "/ITSOWEK/ODPassthru",
                 "0",
                 "",
                 "0",
                 "Loan+Delinquency+Reports+LO",

"v7126-9241-9243-9242-NOB1-1FAAA%24-0-1446-25257-6849-85-79-0-1-0-%5E%0
19040%0110000%011000999",
                 "plugin",
                 "0");
   </script>
   </body>
</html>
```

Example 3-21 shows the ODPassthru servlet code snippet.

Example 3-21 ODPassthru servlet code snippet

```
HttpSession session = request.getSession();

synchronized (session) {
   try {
      ODServer odServer = (ODServer) session.getAttribute("odServer");

      byte[] results =
odServer.viewerPassthru(request.getQueryString());
      OutputStream outputStream = response.getOutputStream();
      outputStream.write(results);
   }
```

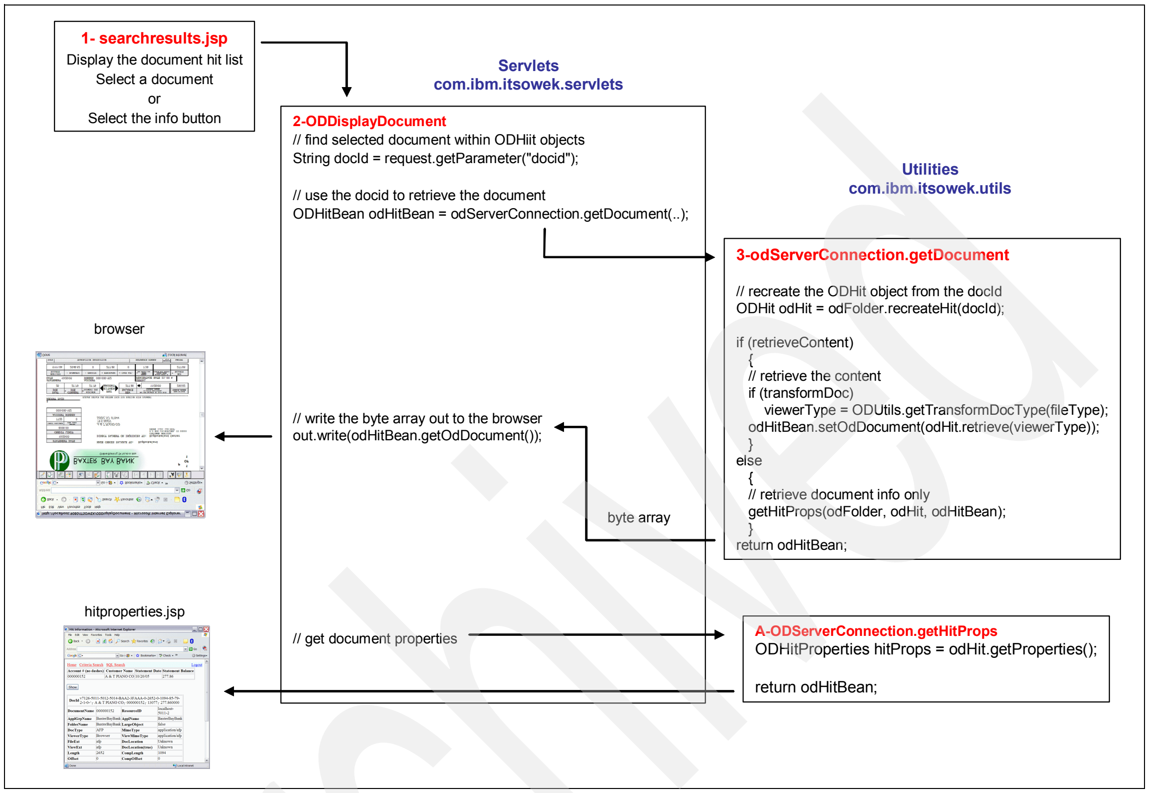
Figure 3-12 shows the document as displayed by using the applet viewer.



```
REPORT   D08100100                  NATIONAL BANK                        DATE  10/01
BANK     001                                                             TIME  16:03
FROM     10/01/08                                                            MODE  9
TO       10/01/08              LOAN DELINQUENCY REPORT                       PAGE 00

   LOAN         LOAN            CUSTOMER           DELINQUENT    DELINQUENT    DELINQU
  NUMBER       AMOUNT             NAME               30 DAYS       60 DAYS       90 DA

0000010000  $10000000.00   MCMULLIGAN, PATRICK   $      50.00  $      50.00  $
0000010001  $    11000.00   ABBOTT, DAVID         $     100.00  $     200.00  $
0000010002  $    12000.00   ABBOTT, DAVID         $     140.00  $        .00  $
0000010003  $    13000.00   ABBOTT, DAVID         $     150.00  $        .00  $
0000010005  $      500.00   ROBINS, STEVEN        $      50.00  $        .00  $
0000010006  $     1000.00   PALMER, ARNOLD        $      75.00  $     150.00  $      22
0000010007  $      650.00   PETERS, PAUL          $      50.00  $        .00  $
0000010008  $     9000.00   ROBERTS, ABRAHAM      $     120.00  $        .00  $
0000010009  $     8000.00   SMITH, RANDOLPH       $     115.00  $        .00  $
0000010010  $     8500.00   KLINE, PETER          $     110.00  $        .00  $
0000010017  $    10000.00   WILLIAMS, ALFRED      $      50.00  $      50.00  $
0000010019  $    11000.00   JAMES, TIMOTHY        $     100.00  $     200.00  $
0000010022  $    12000.00   THOMAS, JAMES         $     140.00  $        .00  $
0000010026  $    13000.00   ROBBINS, KARL         $     150.00  $        .00  $
```

Figure 3-12   Line data viewer applet

Figure 3-13 shows a summary of how the application retrieves and displays a document.



1- searchresults.jsp
Display the document hit list
Select a document
or
Select the info button

2-ODDisplayDocument
```
// find selected document within ODHiit objects
String docId = request.getParameter("docid");

// use the docid to retrieve the document
ODHitBean odHitBean = odServerConnection.getDocument(..);

// write the byte array out to the browser
out.write(odHitBean.getOdDocument());

// get document properties
```

3-odServerConnection.getDocument
```
// recreate the ODHit object from the docId
ODHit odHit = odFolder.recreateHit(docId);

if (retrieveContent)
   {
   // retrieve the content
   if (transformDoc)
      viewerType = ODUtils.getTransformDocType(fileType);
   odHitBean.setOdDocument(odHit.retrieve(viewerType));
   }
else
   {
   // retrieve document info only
   getHitProps(odFolder, odHit, odHitBean);
   }
return odHitBean;
```

A-ODServerConnection.getHitProps
```
ODHitProperties hitProps = odHit.getProperties();

return odHitBean;
```

Figure 3-13 Program flow for retrieving and displaying a document

3.8 Disconnecting from OnDemand

In the previous examples, we show how a user is connected or logged on to Content Manager OnDemand. We also show how the ODServer object that contains the connection is saved in the HttpServletSession object during the entire user experience until the user's session is terminated.

The application must always attempt to disconnect the OnDemand connection either from an explicit user logoff action or when another condition, such as a session time-out, is detected. The examples in this chapter show the application persisting the connection in the session object. Another alternative is to persist and control the connections by connection pooling, which we discuss in Chapter 6, "Connection pooling and connection handling" on page 111.

The ODLogoff servlet is invoked whenever a user selects to log off their OnDemand connection from any of the JSP files. The application should always try to perform the ODServer.logoff() and ODServer.terminate() methods as shown in Example 3-22. These methods clean up user connections and enable the shared libraries to free up allocated native system memory.

Example 3-22 ODLogoff servlet code snippet

```
HttpSession session = request.getSession();

synchronized (session) {
   ODServer odServer = (ODServer) session.getAttribute("odServer");
   try {
      if (odServer != null) {
         odServer.logoff();
         odServer.terminate();
      }
      session.removeAttribute("odServer");
      session.removeAttribute("folderbean");
   }
```

The application might also want to implement an HttpSession listener that is invoked by the application server whenever an HttpSession object becomes invalid. Ensure that the Web.xml is configured for a session listener as shown in the following example:

```
<listener-class>com.ibm.itsowek.utils.ODSessionListener</listener-class>
```

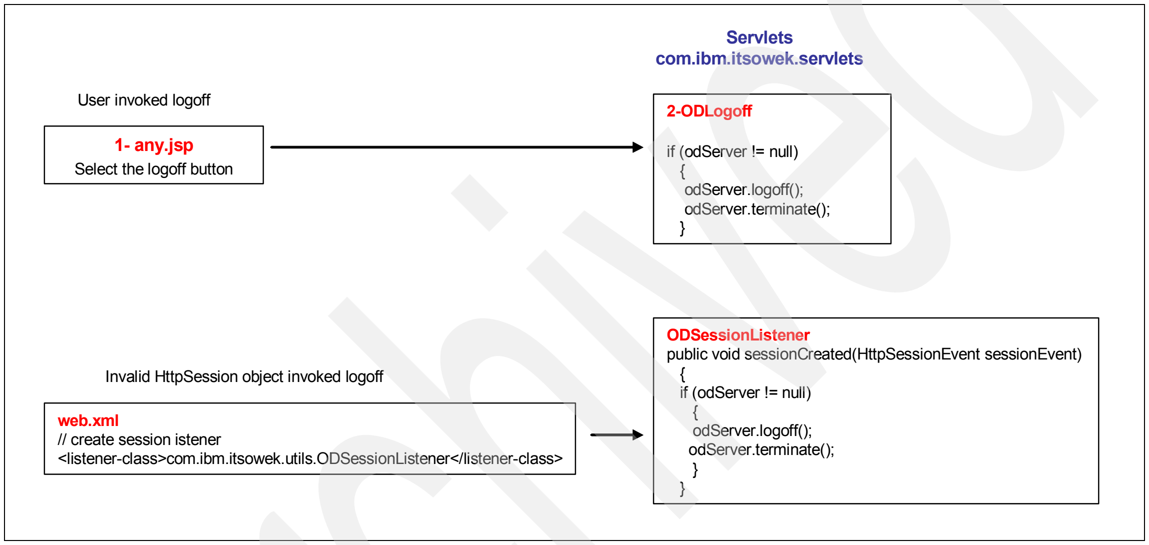
Our example has implemented the HttpSessionListener.sessionCreated() method to terminate the connection and remove any object references from the session object as shown in Example 3-23.

Example 3-23 ODSessionListener.sessionCreated() method code snippet

```
public void sessionCreated(HttpSessionEvent sessionEvent) {
   HttpSession session = sessionEvent.getSession();
   ODServer odServer = null;
   try {
      if (session != null) {
         odServer = (ODServer) session.getAttribute("odServer");
      }
```

```
        synchronized (odServer) {
           if (odServer != null) {
              odServer.logoff();
              odServer.terminate();
           }
        }
     }
```

Figure 3-14 shows a summary of how the application disconnects from the OnDemand server.



```
2-ODLogoff
if (odServer != null)
  {
   odServer.logoff();
   odServer.terminate();
  }
```

```
web.xml
// create session istener
<listener-class>com.ibm.itsowek.utils.ODSessionListener</listener-class>
```

```
ODSessionListener
public void sessionCreated(HttpSessionEvent sessionEvent)
  {
  if (odServer != null)
     {
     odServer.logoff();
    odServer.terminate();
     }
  }
```

Figure 3-14   Program flow for disconnecting from the server

4 Internet use case

In this chapter, we provide a use case of how to use the Java APIs for online Internet access to IBM Content Manager OnDemand.

The chapter covers the following topics:

- Use case overview
- Connection pooling consideration for the use case
- Sample application for the use case

4.1 Use case overview

A common use case across many different business segments involves allowing registered customers to select a range of documents to view. For example, in the banking industry case, these documents can be one or more bank statements that can be selected from a predefined number of months. For health insurance companies, the documents might be explanation-of-benefit statements. For utility companies, the documents might be bills or invoices.

The customer is usually a user who is registered to the company's Web site. The registration process gives entitlement back to the user in the form of credentials, perhaps a user ID and password. To gain access to the company applications, the user submits their credentials from a Web page, such as a portal, to be authenticated by the company's Web application. After the user is authenticated, the application places constraints on what tasks the user can perform when navigating the Web site.

Currently Content Manager OnDemand requires each user who needs access to have a defined user ID. However, it is not manageable to create an OnDemand user ID for each registered Internet user. The user population can reach into the thousands or millions. Therefore, most applications perform a search on behalf of a user by using a user ID that has the permissions to search a specific folder and retrieve one or more documents. The assumption is that if the user is authenticated, and the constraints are placed on the search values to guarantee unique results, then one or more user IDs can be defined to have full permissions to the folder that is searched.

Regardless of the industry that has OnDemand statements to present to the customer, typically two kinds of transactions are possible that a customer can perform.

- Displaying a search results list of documents, usually within a predefined date range.

  The application performs the search on behalf of the user. This type of transaction is in contrast to examples in Chapter 3, "ODWEK Java API examples" on page 53. In that chapter, a user is given permission to select a folder to search and to specify search criteria for one of more folder fields. In this use case, after a user is authenticated, the application chooses a unique key or key combination to ensure that the search results are for that user and performs the search without any user interaction.

- Selecting a document from the search results list that is presented

  The data type of the document comes into consideration when displaying content to Internet users. For example, if the document is stored in Content Manager OnDemand as an AFP data stream, either the users need an AFP viewer program locally installed on their machine, or data conversion must occur to provide a data stream that is more palatable for viewing. In many cases, transforms are performed by the application, such as AFP to PDF, to make it easier for the user to view. The PDF viewer is easily obtainable if it is not already installed on the user's machine. The AFP viewer, from a plug-in for example, might be more difficult to obtain and manage by the company that is providing the Web service.

The following steps are involved for this example:

1. A user signs on to Web site with their credentials.
2. The Web application authenticates the user.
3. The Web application retrieves an OnDemand connection from a pool of connections.
4. The Web application selects a predefined OnDemand folder to search.
5. The Web application assigns one or more OnDemand folder field values to perform a search unique to the user.
6. The Web application performs a search, for example by account number and date range, and returns a search result list to the user.
7. The Web application closes the OnDemand folder.
8. The Web application releases the OnDemand connection back to the pool of connections.
9. The user selects a document to view from the search results list.
10. (Optional) The user is re-authenticated before being allowed to retrieve the document.
11. The Web application retrieves an OnDemand connection from a pool of connections.
12. The Web application retrieves the document and optionally performs a data transformation before releasing the document to the user.
13. The Web application releases the OnDemand connection back to the pool of connections.

4.2 Connection pooling consideration for the use case

For the kind of use cases mentioned in the previous section, consider implementing connection pooling within the design of the OnDemand Web application. While we discuss connection pooling in detail Chapter 6, “Connection pooling and connection handling” on page 111, we explain connection pooling that is applicable in this use case.

The implementation of connection pooling for OnDemand Web access can provide a significant performance improvement when using APIs. To perform an OnDemand search, a user ID must be logged on and a folder must be opened. Also, when a user’s session is completed, log off the user and terminate the OnDemand session. Performing the steps to log on and log off repeatedly by using the APIs can degrade the performance of the Web server.

The examples in Chapter 3, “ODWEK Java API examples” on page 53, do not show connection pooling, but show methods on how to save the ODServer connection in the HttpServletSession object. This gives the sense of persisting the user’s connection throughout their set of transactions. The application is dependent upon detecting a session termination by either the user explicitly selecting a logoff link or by another condition such as a session time-out that invokes the HTTP session listener implementation.

Implementing a connection pool can take on many different designs. While there are several methods to implement connection pooling, the method that is selected is based on a design decision made by the architects. The example in this chapter shows a pool of OnDemand connections that are created automatically during the initialization of a servlet. The connections are retrieved and returned during the doGet and doPost methods. All of the OnDemand connections are terminated after the servlet instance is destroyed. Either the connection pool or the application can own the responsibility of creating and opening the ODFolder object. In our example, the application creates the ODFolder object.

The connections in the pool are represented by ODServer object instances. For internet users, such as banking customers, where all of the statements are accessed from one OnDemand folder, the connection pool can be extended to also open the folder as part of creating the ODServer instance. For applications where the OnDemand folder is selected at random by the user, then it might make more sense to open the folder outside of the connection pooling implementation.

Our example uses an ODConnectionPool class to manage the OnDemand server connections. when the servlet initializes, it calls the ODConnectionPool constructor to create a predefined number of ODServer connections as shown Example 4-1.

Example 4-1 ODConnectionPool code snippet

```
this.server = server;
this.username = username;
this.password = password;
this.myProps = myProps;

this.maxConnections = maxConnections;
this.waitIfBusy = waitIfBusy;

if (initialConnections > maxConnections)
   initialConnections = maxConnections;

availableConnections = new Vector(initialConnections);
busyConnections = new Vector();

for (int i = 0; i < initialConnections; i++)
   availableConnections.addElement(makeNewConnection());
}

...code...
private ODServer makeNewConnection() throws Exception {
   try {
      // get the custom afp2pdf transform properties
      String transformPropsName =
myProps.getProperty("TransformProperties");
      Properties transformProps = new Properties();
      transformProps.load(new FileInputStream(transformPropsName));

      // Build the relative URL for the LineDataViewer Applet directory
      // This would be something as '/ITSOWEK/applets'
      String appletDir = new
StringBuffer().append("/").append(myProps.getProperty("ContextRoot"))
         .append("/applets").toString();


      // set the configuration values from my custom properties
      ODConfig odConfig = new
ODConfig(myProps.getProperty("AfpViewer"), // AfpViewer
           myProps.getProperty("LineViewer"), // LineViewer
```

```
            myProps.getProperty("MetaViewer"), // MetaViewer
            Long.parseLong(myProps.getProperty("MaxHits")), // MaxHits
            appletDir, // AppletDir
            myProps.getProperty("Language"), // Language
            myProps.getProperty("TempDir"), // TempDir
            myProps.getProperty("TraceDir"), // TraceDir
            Integer.parseInt(myProps.getProperty("TraceLevel")), //
trace
            // level
            transformProps); // properties

      // Initialize ODServer object and login
      System.out.println("Making a new ODServer connection in the
pool");
      ODServer connection = new ODServer(odConfig);
      // connection.initialize(configDir, appName);
      connection.initialize(new StringBuffer().append("/")
         .append(myProps.getProperty("ContextRoot"))
         .append("/ODPassthru").toString());

      connection.logon(server, username, password);
      return (connection);
```

As the servlet needs a new connection, it invokes the ODConnectionPool.getConnection() method (Example 4-2) to receive an ODServer object from the pool that either existed or was created when all active connections were occupied.

Example 4-2   ODConnectionPool.getConnection() method code snippet

```
if (!availableConnections.isEmpty()) {
   System.out.println("Getting a connection from the list");
   ODServer existingConnection = (ODServer)
availableConnections.lastElement();
   int lastIndex = availableConnections.size() - 1;
   availableConnections.removeElementAt(lastIndex);

   // If connection on available list is closed (e.g.,
   // it timed out), then remove it from available list
   // and repeat the process of obtaining a connection.
   // Also wake up threads that were waiting for a
   // connection because maxConnection limit was reached.

   if (!existingConnection.isInitialized()) {
      notifyAll(); // Freed up a spot for anybody waiting
      return (getConnection());
   }
   else {
      busyConnections.addElement(existingConnection);
      return (existingConnection);
   }
}
else {
   // Three possible cases:
   // 1) You haven't reached maxConnections limit. So
   // establish one in the background if there isn't
   // already one pending, then wait for
   // the next available connection (whether or not
   // it was the newly established one).
   // 2) You reached maxConnections limit and waitIfBusy
   // flag is false. Throw SQLException in such a case.
   // 3) You reached maxConnections limit and waitIfBusy
   // flag is true. Then do the same thing as in second
   // part of step 1: wait for next available connection.

   System.out.println("No connections avail. Trying to make a new
one");
   if ((totalConnections() < maxConnections) && !connectionPending)
      makeBackgroundConnection();
   else if (!waitIfBusy)
      throw new Exception("Connection limit reached");

   // Wait for either a new connection to be established
   // (if you called makeBackgroundConnection) or for
   // an existing connection to be freed up.

   try {
      System.out.println(" Please wait...");
      wait();
   }
   catch (InterruptedException ie) {
   }

   // Someone freed up a connection, so try again.
   return (getConnection());
}
```

4.3 Sample application for the use case

The sample application is created for the Internet use case that is described in this chapter. When a user requests access to the Internet Web site, the sample application prompts the user for the user's user ID and password (Figure 4-1).

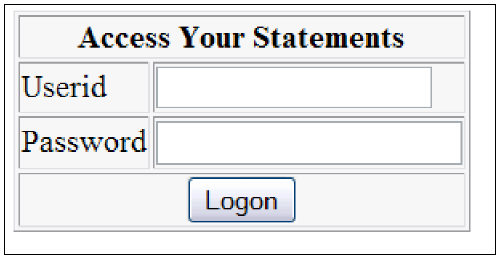


Figure 4-1 Initial login

After the user enters their credentials, an authentication process can occur. We simulated this by calling an ODAuthenticate servlet (Example 4-3). The servlet calls another process to authenticate the user and returns the user's account number in order for our application to perform a search that is unique to the user. The account number is a folder field. When this field is searched by a given value, all of the results are returned for the specified account number value.

Example 4-3 ODAuthenticate servlet code snippet

```
String userid = request.getParameter("userid");
String password = request.getParameter("password");

if (!userid.equals("") && !password.equals("")) {
   try {
      /* authenticate the user credentials such as */
      /* ClientAuthenticate ca = new ClientAuthenticate(); */
      /* String accountNumber = ca.authenticate(userid, password); */
      /* request.setAttribute("accountnumber", accountNumber); */

      // test
      String accountNumber = request.getParameter("testacct");
      request.setAttribute("accountnumber", accountNumber);
      // end test
   }
```

For this example, all statements are defined to one OnDemand folder. Two folder field names are used to perform the searches. Rather than try to discover the folder field names by iterating through the list of ODCriteria objects as we did in

Chapter 3, “ODWEK Java API examples” on page 53, we identify those fields in a properties file to improve performance. The OnDemand administrator must keep the application developer aware of any changes to the folder name or folder field.

After the user is authenticated, the request is forwarded to the ODSearchRetrieve servlet. This servlet performs the OnDemand folder search and retrieves an Ondemand document depending on the request parameters. We added an init() method that invokes the ODConnectionPool constructor to create the initial set of OnDemand connections. See Example 4-4.

Example 4-4 ODSearchRetrieve.init() code snippet

```
try {
   // get the custom properties
   props = new java.util.Properties();
   java.net.URL url =
ClassLoader.getSystemResource("ODWEK.properties");
   props.load(url.openStream());

   String userid = props.getProperty("DefaultUserid");
   String password = props.getProperty("DefaultPassword");
   String server = props.getProperty("DefaultServer");

   connectionPool = new ODConnectionPool(server, userid, password,
props);

   String displayFields = props.getProperty("DefaultDisplayFields");
   if (displayFields != null && !displayFields.equals("")) {
      userDisplayFields = displayFields.split(",");
      // userDisplayFields = Arrays.asList(splitFields);
   }
}
```

The servlet init method is invoked whenever the servlet is first instantiated. We elected to have the servlet loaded during the application server startup as shown in the following Web.xml configuration file:

```
<load-on-startup>-1</load-on-startup>
```

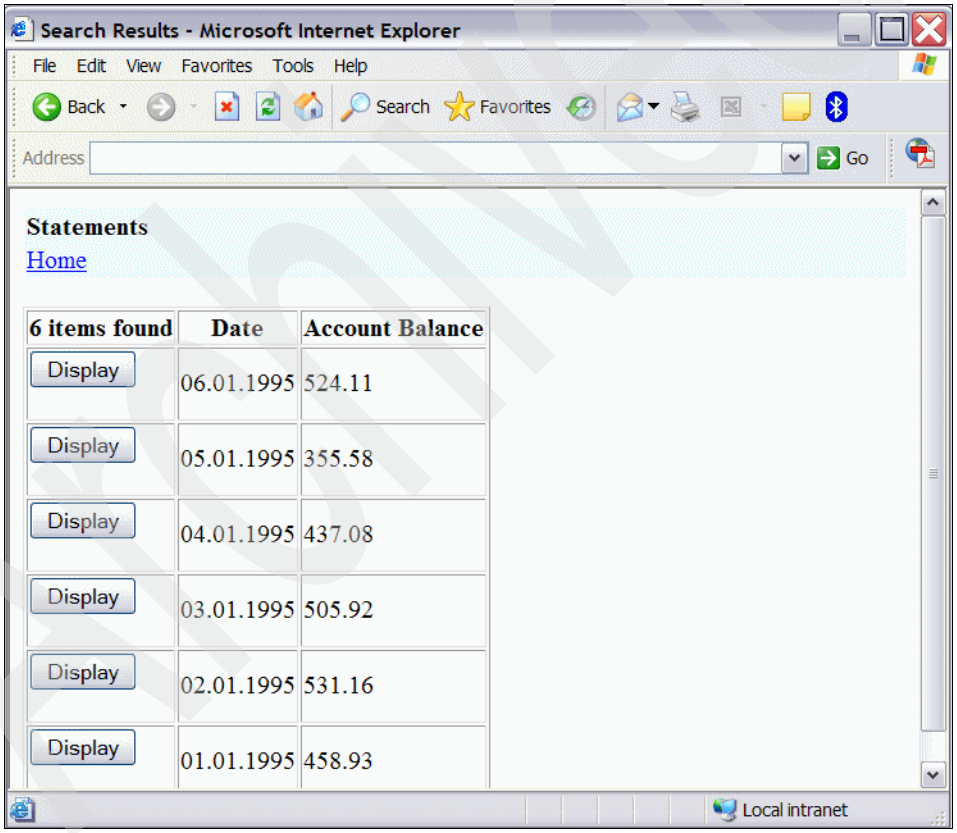
The ODSearchRetrieve servlet performs two functions based on the function request that is forwarded by either the ODAuthenticate servlet or the searchresultsexternal.jsp file. For a search operation, the servlet performs a search by opening a predefined OnDemand folder. The servlet sets the account folder field value and date field values to search for up to a predefined number of previous month’s worth of statements. See Example 4-5 on page 100.

Example 4-5 ODSearchRetrieve servlet code snippet

```
HttpSession session = request.getSession(true);

synchronized (session) {
   ODServer odServer = null;
   ODFolder odFolder = null;
   try {

      // get an OnDemand connection from the pool
      odServer = connectionPool.getConnection();

      String requestFunction = request.getParameter("requestfunction");

      ODServerConnection odServerConnection = new ODServerConnection();
      String folderName = props.getProperty("DefaultFolder");

      if (requestFunction != null && requestFunction.equals("search"))
{
         // get account number
         String keyValue = (String)
request.getAttribute("accountnumber");
         // String keyValue = request.getParameter("keyvalue");

         // get the folder and field names to search
         String searchFieldName =
props.getProperty("DefaultSearchField");
         String searchDateName =
props.getProperty("DefaultSearchDateField");
         odFolder = odServer.openFolder(folderName);
         ODCriteria critKey = odFolder.getCriteria(searchFieldName);
         ODCriteria critDate = odFolder.getCriteria(searchDateName);

         // set the folder search criteria
         critKey.setSearchValue(keyValue);
         critKey.setOperator(ODConstant.OPEqual);

         //calculate previous 6 months begin date
         SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yy");
         Calendar c = Calendar.getInstance();
         String endDate = sdf.format(c.getTime());
         c.add(Calendar.MONTH, -6);
         String beginDate = sdf.format(c.getTime());
         critDate.setSearchValues(beginDate, endDate);
         critDate.setOperator(ODConstant.OPBetween);
         odFolder.setOrSearchCriteria(false);

         // perform search
         ODFolderBean odFolderBean = new ODFolderBean();
         String[] folderFieldDisplayNames = odFolder.getDisplayOrder();

         // set my user defined fields to display if any in the
         // properties file
         if (userDisplayFields == null) {

odFolderBean.setDisplayOrderArray(odFolder.getDisplayOrder());
         }
         else {
            odFolderBean.setDisplayOrderArray(userDisplayFields);
         }

         odServerConnection = new ODServerConnection();
         List<ODHitBean> listHits =
odServerConnection.getSearchResults(odServer, odFolder,
            null, odFolderBean.getDisplayOrderArray());

               .getDisplayOrderArray());
         if (listHits != null) {
            request.setAttribute("hits", listHits);
         }

         // odFolderBean.setDisplayOrderArray(folderFieldDisplayNames);
         session.setAttribute("folderbean", odFolderBean);
         odFolder.close();

request.getRequestDispatcher("/searchresultsexternal.jsp").forward(requ
est, response);
      }
      else {
         if (requestFunction != null &&
requestFunction.equals("displaydoc")) {
            // get request parameters
            String docId = request.getParameter("docid");

            // retrieve the OnDemand document
            ODHitBean odHitBean =
odServerConnection.getDocument(odServer, folderName, docId,
               true, true);
```

```
                response.setContentType(odHitBean.getMimeType());

                // write the byte array out to the browser
                ServletOutputStream out = response.getOutputStream();
                out.write(odHitBean.getOdDocument());
            }
        }
        // release this OnDemand connection back to the pool
        connectionPool.free(odServer);
    }
```

The search results are displayed in the searchresultexternal.jsp file, which is rendered as shown in Figure 4-2.


Search Results - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
Go
Local intranet

Statements
Home

6 items found	Date	Account Balance
Display	06.01.1995	524.11
Display	05.01.1995	355.58
Display	04.01.1995	437.08
Display	03.01.1995	505.92
Display	02.01.1995	531.16
Display	01.01.1995	458.93

Figure 4-2 The rendered searchresultsexternal.jsp page

In this case, the user is presented with a list of statements for the last six months. The user can click the Display button, and the ODSearchRetrieve servlet is invoked, but this time with a request parameter to displaydoc. The servlet requests an OnDemand connection from the connection pool and retrieves the document by using the document ID that passed in from the searchresultsexternal.jsp file. This action might not be desirable for some Web applications. As discussed in Chapter 3, “ODWEK Java API examples” on page 53, another technique is to create a list of ODHit objects that can be maintained in some way on the server. Instead of sending the document ID as part of the search results, the search results JSP sends the relative location of the selected document. The search servlet matches the selected relative location with the ODHit object in the saved list.

In this example, we convert the AFP data stream to PDF by calling the AFP2PDF transformation utility. This is accomplished by either how the ODConfig object properties are set or how the retrieve method is invoked on the ODHit object. When we determine that the file type is AFP, we set the ODHit object viewer to PDF (Example 4-6).

Example 4-6   ODUtil code snippet

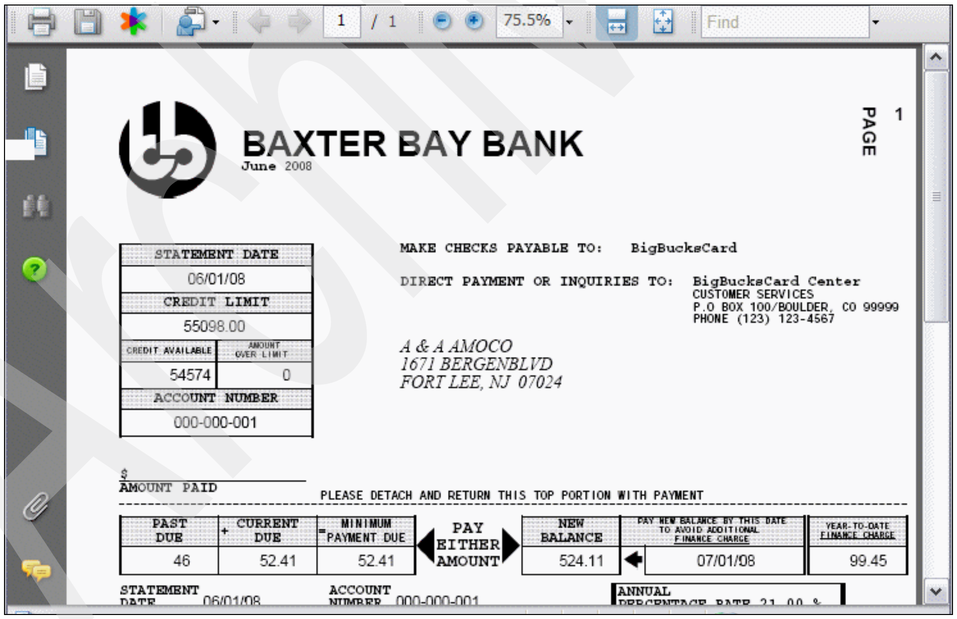
```
String str;
switch (fileType) {
case ODConstant.FileTypeAFP:
   str = ODConstant.PDF;
   break;
case ODConstant.FileTypeLINE:
   str = ODConstant.APPLET;
   break;
default:
   str = ODConstant.NATIVE;
   break;
}
return str;
```

By setting the object viewer to PDF, we tell Content Manager OnDemand to transform the AFP document to PDF by using the AFP2PDF Transform utility (Example 4-7). When the document is successfully retrieved, we get the MIME type of the document.

Example 4-7 ODServerConnection.getDocument() method code snippet

```
// retrieve the content
if (retrieveContent) {
   // retrieve the content
   char fileType = odHit.getDocType();
   String viewerType = "";
   if (transformDoc) {
      viewerType = ODUtils.getTransformDocType(fileType);
   }
   odHitBean.setOdDocument(odHit.retrieve(viewerType));
}
```

Figure 4-3 shows the results of the AFP document being transformed and displayed as a PDF document by using Adobe® Reader.



BAXTER BAY BANK
June 2008

PAGE 1

STATEMENT DATE
06/01/08
CREDIT LIMIT
55098.00

CREDIT AVAILABLE	AMOUNT OVER LIMIT
54574	0

ACCOUNT NUMBER
000-000-001

MAKE CHECKS PAYABLE TO: BigBucksCard

DIRECT PAYMENT OR INQUIRIES TO: BigBucksCard Center
CUSTOMER SERVICES
P.O BOX 100/BOULDER, CO 99999
PHONE (123) 123-4567

A & A AMOCO
1671 BERGENBLVD
FORT LEE, NJ 07024

$
AMOUNT PAID

PLEASE DETACH AND RETURN THIS TOP PORTION WITH PAYMENT

PAST DUE	+ CURRENT DUE	= MINIMUM PAYMENT DUE	PAY EITHER AMOUNT	NEW BALANCE	PAY NEW BALANCE BY THIS DATE TO AVOID ADDITIONAL FINANCE CHARGE	YEAR-TO-DATE FINANCE CHARGE
46	52.41	52.41		524.11	07/01/08	99.45

STATEMENT DATE 06/01/08 ACCOUNT NUMBER 000-000-001 ANNUAL PERCENTAGE RATE 21.00 %

Figure 4-3 AFP document transformed to PDF

Part 2

Best practices, hints, and tips

In this part, we discuss best practices, hints and tips on ODWEK Java API usages with advanced topics such as connection pooling, globalization, and searching.

Multicultural support: For more information about multicultural support, see the definition in 7.1, “Globalization overview” on page 144.

This part includes the following chapters:

5

Introduction to best practices, hints, and tips

In this chapter, we provide a quick overview of the content of the remaining chapters in the book. We introduce the topics and areas where best practices, hints, and tips are discussed in these chapters. We provide a synopsis of each chapter's content in the following sections. Where appropriate, topics of a more sophisticated nature are indicated as advanced.

An OnDemand Web Enablement Kit (ODWEK) Web application can often be written without employing these advanced techniques. However, an application that operates under heavy load, such as a middle-tier component or a batch process, can employ one or more of these advanced methods to achieve the required performance and scalability.

Chapter 6, “Connection pooling and connection handling”

By using *connection pooling*, applications, especially middle-tier and portal applications, can scale up to meet the demands of large numbers of concurrent requests to IBM Content Manager OnDemand. A connection pool accomplishes this by allocating only the number of connections that are permitted for simultaneous access to the server and queuing the excess traffic, rather than allocating a separate connection immediately for each new request. Connection pooling saves the time that is required to initialize new connections and prevents a flood of requests from overwhelming the application server.

In this chapter, we provide an overview of connection pooling and best practices with regard to the ODWEK Java API implementation. We also discuss the best practices for developing connection pools using the ODWEK Java APIs.

Resource allocation and memory problems occur in applications that fail to open and close connections to the OnDemand server in an appropriate manner. We also provide a detailed understanding of the allocation of resources by Content Manager OnDemand connections and explain how to create and terminate connections properly.

Chapter 7, “Globalization”

Many OnDemand servers store documents in multiple languages and serve clients with different languages, locales, and character sets. To ensure successful implementation of the OnDemand system, you must understand and take into consideration the usage of different encoding schemes and conversions.

In this chapter, we address code page conversion and other data-access considerations that arise in heterogeneous environments. We also discuss ICU, which is the multicultural support API shipped with ODWEK.

Multicultural support: For more information about multicultural support, see the definition in 7.1, “Globalization overview” on page 144.

Chapter 8, “Folder searching”

Locating documents that are archived by Content Manager OnDemand is one of the core functions of any ODWEK Web application feature. Understanding and proper usage of the search techniques is critical in the success of your Web application. Document searches can be accomplished by performing either a criteria search or an SQL search.

In this chapter, we provide an overview of the techniques for performing OnDemand document searches by using the ODWEK Java APIs.

Chapter 9, “Document retrieval”

After documents are located, they can be retrieved and displayed to users. Documents are stored in Content Manager OnDemand either as a unit or in segments (for OnDemand large documents), in which case the segments are retrieved sequentially.

In this chapter, we discuss how documents are retrieved.

Chapter 10, “Applets, plug-ins, and transforms”

Retrieved documents are usually converted from their original storage format into a format that is more convenient for clients, such as to convert AFP documents that are generated by mainframe processes into PDFs, which are easily displayed on a PC client. Options for document conversion include utilities that are bundled with ODWEK and third-party and custom solutions.

Note: These options require separate entitlement.

In this chapter, we explain how the plugs-ins and transforms are used in document retrieval. We pay special attention to AFP resources and large objects.

Chapter 11, “Document storing and updating”

Documents are nearly always loaded to the OnDemand server by an administrator or batch process by using the command-line utilities that are included with Content Manager OnDemand. However, there is an occasional need to store documents programmatically. The ODWEK Java APIs also accommodate the occasional need to update or delete document index information on the server.

In this chapter, we discuss these APIs and address care that you must take when performing server updates using the APIs.

Chapter 12, “Memory and performance”

The ODWEK Java APIs can be employed in any architectural tier and embedded virtually anywhere that a Java application can run. As such, performance and resource tuning is a platform-centric, rather than ODWEK-centric, effort.

In this chapter, we present investigation and tuning techniques that we have found to be useful. We also offer guidelines for memory allocation and platform tuning for ODWEK applications.

Chapter 13, “Troubleshooting”
In this chapter, we present general troubleshooting techniques and make recommendations for addressing some of the most commonly-reported problems.

6 Connection pooling and connection handling

In this chapter, we provide an overview of connection pooling and best practices in regard to the OnDemand Web Enablement Kit (ODWEK) Java API implementation. We also discuss best practices for developing connection pools by using the ODWEK Java APIs.

The chapter includes the following topics:

- Connection pooling overview
- Connection pooling objects and pooling technique
- A simple connection pool code example
- Thread safety
- Resource consumption control
- Timeout

6.1 Connection pooling overview

When a client application (or a Web application in the context of this book) connects to an IBM Content Manager OnDemand (OnDemand) server, it establishes and consumes several resources in order to retrieve the document data that is being requested by the user. A typical sequence of requests to retrieve a document is to log on, retrieve the folder list, open a folder, issue a query, and then retrieve the document from the document hitlist. All of these requests consume resources in terms of mid-tier memory and CPU, OnDemand server memory and CPU, network, and elapsed time.

Connection pooling allows for the reduction of resource consumption through the re-use of objects by multiple incoming connections (user requests). Therefore, the use of connection pooling improves performance.

For example, consider a Web-based Internet application, such as customer statement presentation, where thousands of customers access their statements on an hourly basis. For each customer access, a logon, folder open, folder query, and document retrieve request must be performed. In this scenario, all customers can use a single logon ID (to the OnDemand server) and open a single folder (for example monthly statements). The query against the folder is different for each customer (for example, based on the social security number) and thus results in different document data being returned to each customer.

If we do the same logon and the same folder open thousands of times per hour, it is more efficient to perform these requests only a few times and store the results in a pool that can be re-used over and over again.

6.1.1 Benefits of connection pooling

The pooling of connection resources results in multiple benefits both on the Web server application side and on the OnDemand server side. The benefits include a reduction in memory and CPU consumption and an improvement in overall response time and network throughput:

- Connection pooling helps to reduce memory consumption.

  From the Web server application perspective, because the pooled objects are shared among the connected users, it is only necessary to maintain a pool as large as the maximum number of concurrently active users. From the OnDemand server perspective, the order of magnitude is less than the number of users who are expected to log onto the system. That is, the maximum number of concurrently active users, which might be hundreds or thousands of users, is much less than the number of all users who use the system, which might be thousands or tens of thousands of users.

- Connection pooling helps to reduce CPU consumption.

  The Web server issues fewer requests since the pooled objects are already available. The processing of fewer requests results in a reduction in CPU consumption. The OnDemand server also receives fewer requests and thus consumes fewer CPU resources.

- Connection pooling helps to improve response time.

  Accessing the pooled objects directly on the Web server is much faster than retrieving another copy of the objects from the OnDemand server. When a request is sent to the OnDemand server, it executes faster because fewer requests are issued against the server.

- Connection pooling helps to reduce network traffic.

  Reduced network traffic is the result of the reduced number of requests and responses that are sent over the network. The reduction in network traffic can also lead to improved network throughput.

6.2 Connection pooling objects and pooling technique

You can develop code that pools a variety of Content Manager OnDemand objects. The code requirements vary in sophistication based on the needs of the application. In this section, we discuss the Content Manager OnDemand objects and a pooling technique that uses arrays.

6.2.1 The ODServer class

The ODWEK Java APIs are implemented as a set of classes. For a set of requests issued by a single user, the various classes are instantiated to form a group of objects.

The main class that represents an OnDemand session is the ODServer class. Each ODServer class can handle a connection and a login to a single OnDemand server.

The ODServer class provides five important methods for connection and session handling:

- constructor

  All configuration data is passed to the constructor. No other method exists to change the configuration data. The ODConfig object is instantiated within the constructor and acts as a container for the configuration data for the specific session.

- initialize

  The ODServer.initialize() method must be called before establishing a connection to an OnDemand server. If the ODServer.initialize() method is not called, the ODServer object cannot perform a logon to an OnDemand server.

  After the ODServer object is initialized, you can perform as many logon and logoff requests as required. You do not need to terminate and initialize the ODServer object for each login. You can determine if an ODServer object has been initialized by using the isInitialized() method.

  Each time an ODServer is initialized, an internal counter is incremented. By incrementing the internal counter, the ODWEK Java APIs can keep a count of how many client applications are running and how long the shared resources must be kept in memory.

- logon

  The ODServer.logon() method establishes the TCP/IP connection with, and logs onto, the specified OnDemand server. After the completion of a successful logon, a token for the session is created on the OnDemand server and transferred back to the logon API. This token is used throughout the client connection as a session identifier.

  The logon() method does not use significant memory resources.

- logoff

  Locally, the ODServer.logoff() method closes any open folders (ODFolder) and clears the cached hitlist if one exists. The ODServer.logoff() method terminates all non-shared session-related objects. It has no effect on shared resources.

- terminate

  ODServer.terminate() causes the internal counter, which indicates the number of clients connected to the OnDemand server, to be decremented. When the counter reaches zero, which implies that all clients have terminated their ODServer objects, then the API knows that it is time to release the shared resources from memory. Specifically, “shared resources” refers to AFP resource data.

6.2.2 ODWEK Java API objects and threads

The ODWEK Java API objects provide access to servers, folders, criteria, and hits. These objects are unique to a connection and must not be shared across connections.

Figure 6-1 illustrates the relationship between threads, methods, and objects for two cases, batch and interactive cases. There are many different interaction possibilities and flow scenarios that can result in different thread allocations.

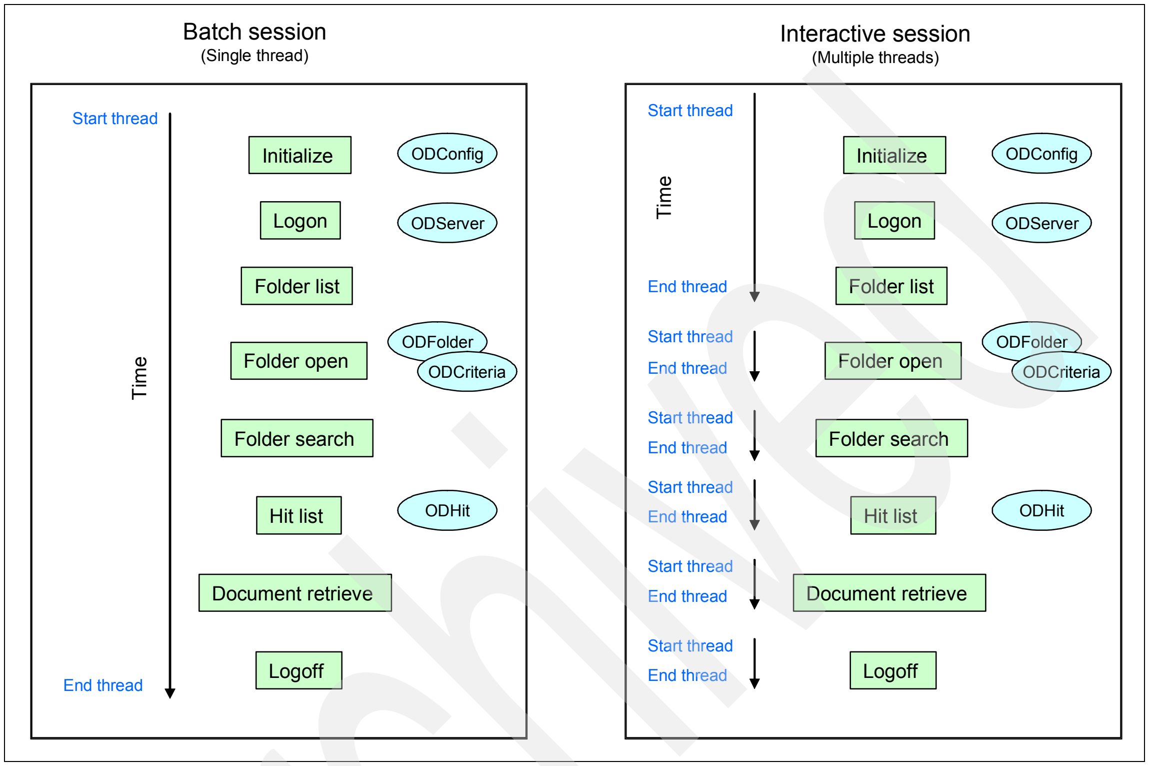


Figure 6-1 Content Manager OnDemand threads, methods, and objects (simplified)

The left side of Figure 6-1 shows the batch case. In this case, a batch program issues the API requests sequentially, logon through logoff. All the requests run on a single thread. All of the objects are created as needed and are destroyed when the program terminates.

The right side of the diagram in Figure 6-1 illustrates the interactive case, which is also known as the *conversational case*. This case can be thought of as a Web server application that supports a set of pages in a browser through which users can enter their requests and view the retrieved data.

When a user enters a user ID and password on the logon page, a servlet is invoked on its own thread in the Web server. The servlet calls both the initialize and the logon methods, which results in the creation of two objects, ODConfig and ODServer. By using the ODServer object, a folder list is obtained and downloaded to the browser for display, at which point, the Web server thread is terminated, and the objects are removed. The user then selects a folder from the folder list to open. A new thread is invoked on the Web server, the folder open method is executed, and the ODFolder and ODCriteria objects are created. A Web page that contains the folder key information is created and downloaded to the browser. The Web server thread is terminated, and the objects are removed.

This process continues throughout the user's session. Between thread invocations, data might need to be saved to be used in the next method or thread invocation. This data needs to be saved in the Web server application that you develop.

Figure 6-2 shows an extension of the conversational scenario presented in Figure 6-1 on page 115. It illustrates the interactions of three users who concurrently access the Web server. As the amount of concurrent user access increases, the number of threads increases, and the number of objects that are required to be stored on the Web server increases. By using connection pooling, you can reduce the number of times that the Web server must make requests from the OnDemand server for the pooled objects.

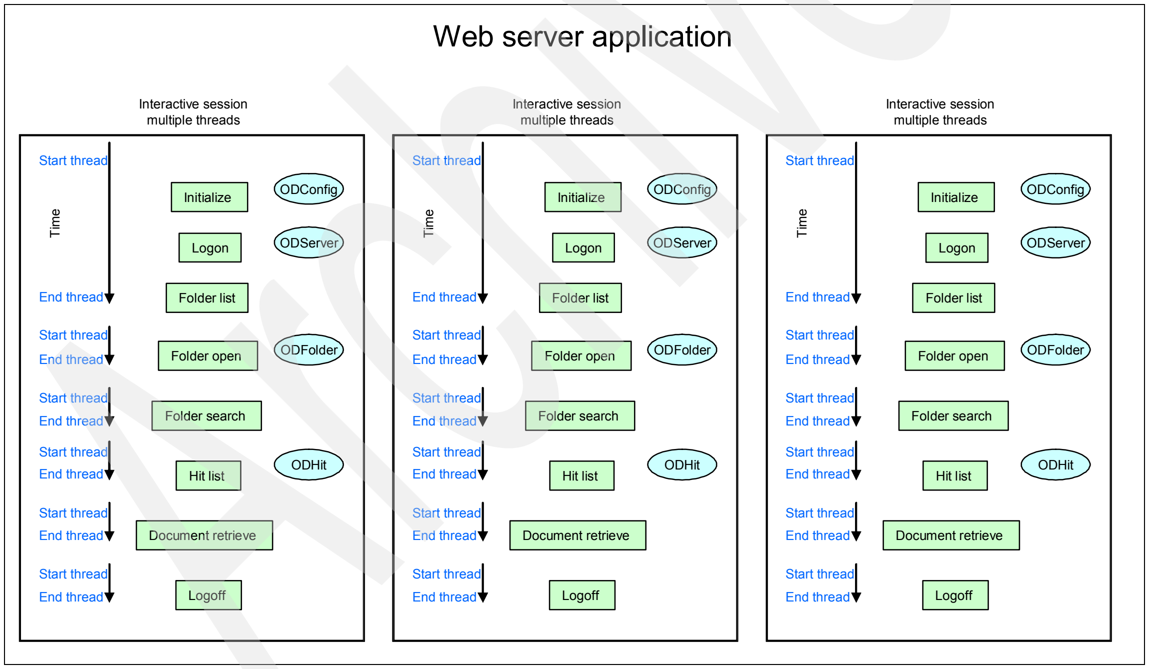


Figure 6-2 Three Content Manager OnDemand concurrently active users

6.2.3 The ODWEK Java API pool levels

The concept of connection pooling entails locally storing objects that can be used by multiple users, thus avoiding continuous requests for the same objects from the server. When looking at the ODWEK Java API objects, in general, there are three connection pool usage scenarios, which are summarized in Table 6-1.

Table 6-1 Connection pool usage scenarios

Initial object stat	**Shared object**	**Usage scenario**
initialize	ODServer	In an intranet application where there are tens of thousands of users each of whom has their own Content Manager OnDemand IDs and permissions. Using connection pooling reduces memory consumption.
inititialize logon	ODServer	In an intranet or Internet application where you can map your Web users to a single Content Manager OnDemand ID.
initialize logon folder open	ODFolder	In an intranet or internet application where you can map your Web users to a single Content Manager OnDemand ID and they can all access the same folder.

The most basic connection pooling level is at the ODServer object level. The ODServer object is created and initialized and is common to all types of user access to the Java APIs. Each user must then be logged on and off between transactions. Pooling at this level is beneficial in the intranet case where there are have tens of thousands of users, each of whom has their own Content Manager OnDemand IDs and permissions.

If the ODServer objects in the pool are used in an environment where you can map multiple users to a single user ID, then these objects can be initialized and logged on. Rather than logging off the user between requests, the ODServer object stays logged on between requests. After the user request is completed, the ODServer object is returned to the pool and is available for subsequent requests.

If the ODServer objects in the pool are used in an environment where you can map multiple users to a single user ID and where users all access the same folder (online billing, as a prime example), then all ODServer objects that are stored in the pool should have their folder left open. The application developer must know the folder name and be able to obtain the ODFolder without re-opening the folder. In this case, the application developer must collect and store the folder name somewhere, for example, in the session object.

6.3 A simple connection pool code example

There are many different designs for implementing connection pooling and many different levels (object types, states) at which objects can be pooled. This example is a simple connection pool at the ODServer object level that uses arrays to store the pooled objects. Storing the pooled data objects is not limited to arrays but can be accomplished by using any kind of storage mechanisms including vectors, hash tables, or both.

Important: The example code presented in this section is not intended for use. Its purpose is only to illustrate the concepts of connection pooling.

6.3.1 The pooling mechanism

This example implements connection pooling by using two arrays. The first array is used to store the ODServer pooled objects and a second "parallel" array is used to store the State of the ODServer objects. Figure 6-3 illustrates these two arrays diagrammatically.

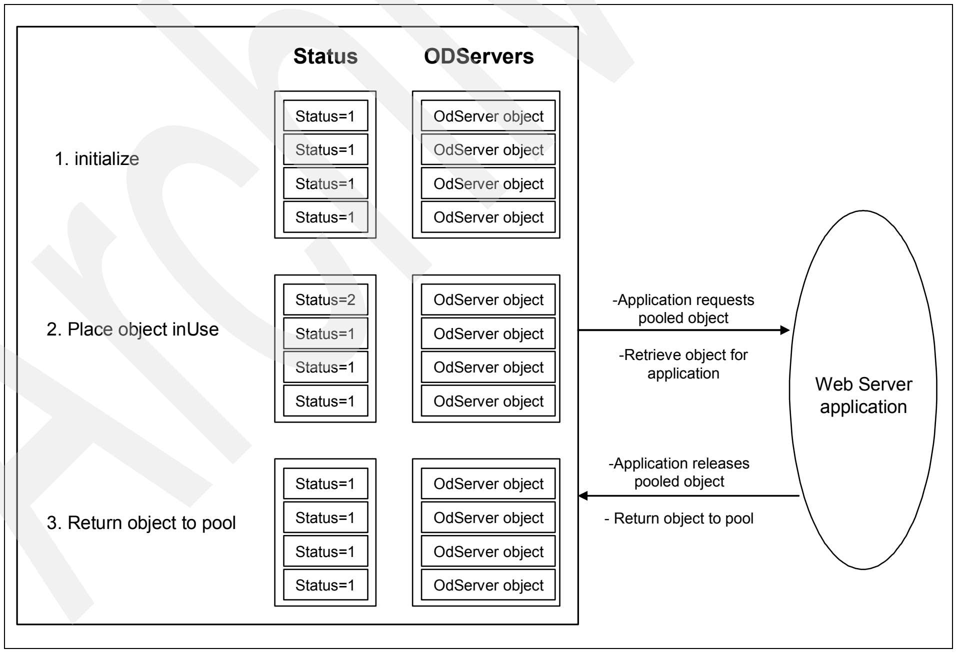


Figure 6-3 Object caching using arrays

When the connection pool class (ODPool) is instantiated, a predetermined number of ODServer objects are created, initialized, and logged on to the OnDemand server. These objects are then stored in the ODServers array. Additionally, a second array (the Status array) is created in which the status of these objects is stored. The initial status value for all of the created objects is set to 1. This indicates that the objects were created and are available for use by the Web server application program.

When the Web server code requests an ODServer object from the pool, the ODServer object is retrieved from the ODServer array and then forwarded to the Web server application. The status of the object in the Status array is changed from a 1 to a 2 indicating that the object is in use. The pool code searches for an available object from the beginning of the array. Thus the objects at the beginning of the array are used more frequently than the objects toward the end of the array.

When the Web server application is finished using the ODServer object, the application returns the ODServer object to the pool. The connection pool code then changes the ODServer status for that object from 2 to 1, indicating that the object is now available for reuse.

The use of arrays is one of multiple mechanisms that are available for object storage. Other collection storage methods, such as vectors or hash tables, are also feasible.

6.3.2 Connection pool code functions

Figure 6-4 illustrates the flow of the connection pool code.

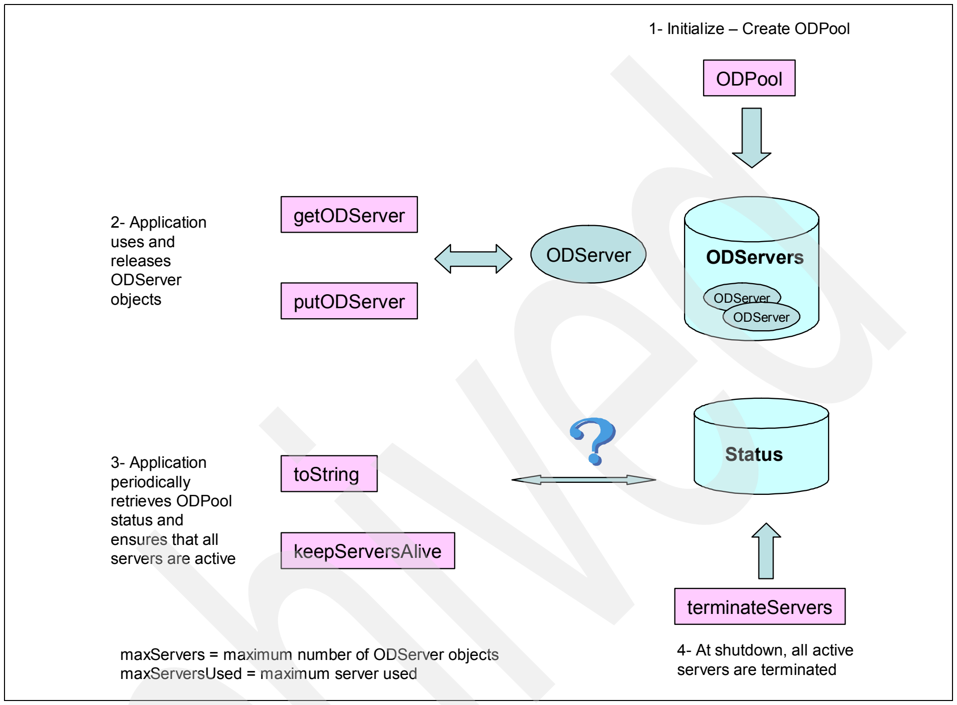

1- Initialize – Create ODPool
ODPool
2- Application uses and releases ODServer objects
getODServer
putODServer
ODServer
ODServers
ODServer
ODServer
3- Application periodically retrieves ODPool status and ensures that all servers are active
toString
keepServersAlive
Status
terminateServers
maxServers = maximum number of ODServer objects
maxServersUsed = maximum server used
4- At shutdown, all active servers are terminated

Figure 6-4 Connection pooling code flow

The connection pool code provides the following functions:

- Pre-allocate and initialize the connections.

  This function is accomplished in the constructor. The ODServer objects are initialized, logged on, and placed in an array that is available for use.

- Get a connection.

  This function is implemented in the getODServer() method. If an ODServer object is not available (meaning that the maximum pool size has been reached), then the ODPool code throws an exception. Otherwise, the ODPool code searches for an available ODServer object and checks to see that it is initialized. If the object is not initialized, then it re-initializes it and forwards the object to the Web application. If the object is initialized, it is immediately forwarded to the Web application.

- Release a connection.

  This function is implemented in the putODServer() method. The ODServer object is returned to the pool. The ODPool code then changes the Status array value to indicate that the ODServer object is now available for reuse.
- Keep the pool connections alive.

  This function is accomplished by polling the OnDemand server from each of the ODServer objects at predefined intervals. Alternatively, connections can be prevented from timing out by setting the OnDemand server timeout period to "Never time out." Pooled connections do not timeout unless some mechanism that is external to the ODPool code forces the time out.
- Get the current ODPool status.

  The toString() method returns the current pool information including the maximum used and currently used server objects.
- Shut down the connection pool.

  The terminateServers() method closes down all the server connections prior to terminating the ODPool code.

6.3.3 Connection pool code sample

For the connection pool code, we show code snippets for the following methods:

- ODPool(): The constructor.
- getODServer(): Obtains an ODServer from the pool.
- putODServer(): Returns an ODServer to the pool.
- keepServersAlive(): Reconnects any timeout servers.
- toString(): Returns information about the current pool status.
- terminateServers(): Terminates the servers and shuts down the connection pool.

The code must include a reference to the package that contains the ODApi.jar file:

```
package com.ibm.edms.od;
```

The ODPool() method: The constructor

The ODPool class is called from the Web server application when it is ready to start connecting to the OnDemand server, which is usually toward the end of its initialization process. The example ODPool constructor accepts the pool initialization parameters as listed in Table 6-2 on page 122.

Table 6-2 Connection pool initialization parameters

Variable	Description
String server	Name or IP address of the OnDemand server to connect to
String username	User ID with correct privileges to service all users requests
String password	Password of specified user name
int portNum	OnDemand server port to connect to
int maxServers	Number of ODServer objects to place in the pool

The ODPool constructor creates two arrays:

- The ODServers array with a size of maxServers
- The Status array with a size of maxServers

The constructor creates a new ODServer object. It then initializes the object and performs an odServer.logon() method to log on to the OnDemand server. After the successful logon, the object is stored in the ODServers array. The ODServer object's status is updated to *Available* in the Status array, and the number of available servers (availableServers) is incremented by one.

Any method that uses the ODServer object in the future does not have to go through the initialization and server logon process again.

Example 6-1 shows the code snippet for the ODPool() constructor method.

Example 6-1 The ODPool() constructor method

```
public ODPool(String server, String username, String password, int
portNum, int maxServers) throws Exception
    {
    //Setup ODServer login/init needs
    this.server    = server;
    this.portNum   = portNum;
    this.ODrunName = "ODPoolTest";

    this.username  = username;
    this.password  = password;

    this.maxServers = maxServers;

    Status = new int[maxServers];
    ODServers = new ODServer[maxServers];

    availableServers =0;
```

```
try
 {
 /* place the ODServer objects in the array */
 for( int i=0; i<maxServers; i++ )
   {
   // create the odConfig object
   ODConfig odConfig = new ODConfig();

   // create a server object
   ODServer odServer = new ODServer(odConfig);
   odServer.initialize(ODrunName);
   odServer.setPort(portNum);

   // logon to the CMOD server
   odServer.logon(server, username, password);
   ODServers[i] = odServer;

   // update the Status array to indicate server available
   Status[i]=1;   // 1 = server available

   // increment the number of available servers in the pool
   availableServers ++;
   }   // end for (int i=0
}

catch(ODException e)
   {
   throw new ODException("createPool error: " + e);
   }

 }  // end constructor ODPool
```

The getODServer() method

The getODServer() method is called by the Web server application program to get an ODServer object from the ODPool. The getODServer() method is listed in Example 6-2 on page 124.

When the getODServer() method is called, it checks to see if an object is available in the ODPool. If an object is available, the getODServer() method copies that object and checks to see that the object is still connected to the OnDemand server. If the object is still connected, the method passes the object to the Web application. Otherwise it re-connects the object, updates the ODServers array, and then passes the ODServer object to the Web application.

If there are no available ODServer objects, the getODServer() method throws a “Connection limit reached” exception.

Example 6-2 The getODServer() method

```
public synchronized ODServer getODServer() throws Exception
    {
    if (availableServers > 0)
      {
      // a server is available, So find available server and return it
      // search for non-busy server
      for (int i=0; i<maxServers; i++)
        {
        if (Status[i] == 1)   // server is available for use
          {
          Status[i] = 2;   // server is no longer available
          odServer = ODServers[i];
          if ( (i+1) > maxServersUsed)
            maxServersUsed = i+1;
          availableServers--;
          totServersUsed++;

          // check to see that odserver is still connected
          if (!odServer.isInitialized())
            {
           try
              {
              // logoff then logon
              odServer.logoff();
              odServer.logon(server, username, password);
              }
           catch (ODException e1)
              {
              throw new ODException("getODServer error: " + e1);
              }
            // place ODServer back in array
            ODServers[i] = odServer;
            }
          }  // end if (Status[i]
        }  // end for (int i=0
      return odServer;
      }
    else
      {
      // there are no servers available
```

```
     System.out.println("No Servers available.. need to specify higher
maxServers");
      throw new Exception("Connection limit reached");
      }
    }  // end method getODServer
```

The putODServer() method

The putODServer() method (Example 6-3) is called by the Web server application program to return an ODServer object to the ODPool. When the putODServer() method is called, it locates the ODServer object in use and changes its status to available. It also increments the number of availableServers.

Example 6-3 The putODServer() method

```
public synchronized int putODServer(ODServer currentServer)
    {
        int retVal = 1; // assume the worst
        for(int i = 0; i < maxServers; i++)
        {
         if (Status[i] == 2)
           {
            ODServer server = ODServers[i];
            if(server.equals(currentServer))
            {
                Status[i] = 1; // set the server to be available to use
                retVal = 0;
                break;
            }
          }
        }
        return retVal;
    } // end method putODServer
```

The keepServersAlive() method

The keepServersAlive() method (Example 6-4) can be called periodically to ensure that all the server connections are still valid.

Example 6-4 The keepServersAlive() method

```
public synchronized int keepServersAlive() throws Exception
    {
    // make sure all non-busy servers are alive
    for (int i=0; i<maxServers; i++)
      {
      if (Status[i] == 1)   // server is available for use
        {
        odServer = ODServers[i];

      try
         {
         // check to see that odserver is still connected
        if (!odServer.isInitialized())
          {
          // logoff then logon
          odServer.logoff();
          odServer.logon(server, username, password);
          }
        else
          {
          // just keep the server alive
          odServer.keepServerAlive();
          }
        // place ODServer back in array
        ODServers[i] = odServer;
         }
      catch(ODException e2)
         {
         throw new ODException("keepServersAlive error: " + e2);
         }
      }  // end if (Status[i]
      }  // end for (int i=0

    } // end method keepServersAlive
```

The toString() method

The toString() method (Example 6-5) returns information about the ODPool including the user name, current ODServer objects in use, maximum number of ODServer objects in use, and size of the ODPool (maxObjects). maxObjects is the maximum number of concurrent connections, which is the same as the maximum number of ODServer objects.

Example 6-5 The toString() method

```
public synchronized String toString()
    {
    String info =
      "ODPool(" + server + "," + username + ")" +
      ", pool Size =" + maxServers +
      ", Currently used=" + (maxServers-availableServers) +
      ", Currently unused=" + availableServers +
      ", Max Concurrently Used=" + maxServersUsed +
      ", Total Servers Used=" + totServersUsed;

    return(info);
    }  // end method toString
```

The terminateServers() method

The terminateServers() method (Example 6-6) is used to force a termination of all existing connections, after which it shuts down the connection pool. This method also forces the garbage collection of all the ODServer objects that were used.

Example 6-6 The terminateServers() method

```
public synchronized void terminateServers()
    {
    for(int i=0; i<maxServers; i++)
      {
      odServer = ODServers[i];
      if (odServer.isInitialized())
        {
        odServer.logoff();
        odServer.terminate();
        }
      }  // end for(int i=0;
    System.exit(0);
    }  // end method terminateServers
```

Important: The code snippets shown in this section provide a simple implementation of an ODPool. The code is not intended to be used as is. You can use the concepts presented and expand on these methods as needed for your environment.

By setting the appropriate pool size, you can ensure optimum usage of connection pooling in your environment.

6.4 Thread safety

Running in a multithreaded environment has become a feature of modern programming architectures. For example, Web applications that run on Web application servers, such as the WebSphere Application Server, automatically use multiple threads. Each call to a Java servlet can be executed on a different thread.

The ODPool class must support both multithreading and object synchronization. You must fully understand these two concepts and implement them within your Web server application.

As illustrated in Figure 6-1 on page 115 and Figure 6-2 on page 116, running the Java APIs in a multi-user environment results in the creation of many threads and many objects within the ODWEK Java APIs and the Web server application code. The number of concurrent threads and in-memory objects varies depending on the design and usage pattern of the system. However, it is safe to say that there will be many of each. To ensure the proper operation of the system, you must ensure thread safety.

Thread-safe object: An object is said to be "thread safe" if the object's state is always valid in a multithreaded environment.

Thread safety exists if an object's state is always valid in a multithreaded environment. This means that an API should not cause data corruption when it is called from multiple threads.

From a practical perspective, to make our objects thread safe, we begin by designing our classes so that all the instance variables are private and the methods that handle object state are *synchronized*.

6.4.1 Instance variables

The Java virtual machine (JVM™) keeps local variables, method parameters, and return values on the Java stack, and each thread has its own stack. Therefore, no thread can harm another thread's stack. This is what we call *thread-safe*.

Instance variables and class variables are only thread-safe if they are private. By making them private, you can control access to the data by controlling access to the code that manipulates the data.

Read-only data does not need to be private. For example, constants, such as static final variables, do not change. They are read only, and therefore, cannot be corrupted by multiple threads that access them at the same time.

Making instance variables private is sufficient in a single threaded environment, but it may not be sufficient in a multithreaded environment. The problem is that the JVM may interrupt the thread that is executing on an object and allow another thread to run against that same object. By the time both threads finish execution, the state of the object is unknown (at best). In such a case, to prevent state corruption, each thread must have its own object instance.

The ODWEK Java APIs are no exception. The APIs are grouped into classes of objects. The object instance must only be accessed by a single thread at a time. Therefore, each thread instance has its own object instance for all the Java API objects.

6.4.2 Synchronization

A critical section of code is a method or a block of code that must be executed atomically as a single, indivisible operation. By declaring the method to be synchronized by using the synchronized keyword, only one thread at a time is allowed to execute that method. By using synchronized objects, you can use wait() and notify() methods to get threads to cooperate in achieving some common goal.

Synchronization involves a performance penalty. Generally speaking, synchronizing a method causes its invocation to be slower because of the blocking and unblocking of threads. In severe cases, blocking and unblocking can cause a deadlock that leads to program delays or hangs.

Rule of thumb: Synchronize only the methods that require synchronization. *Do not* synchronize any other methods. Implementing synchronization in this manner (only as needed) makes your code safe, minimally effects code performance, and most importantly ensures that the code works correctly.

6.4.3 Implementing synchronization in the ODWEK Java APIs

The ODWEK Java APIs support multithreading, which allows multiple concurrent connections to be established and maintained with an OnDemand server. The multiple threads allow for the creation of multiple ODServer objects and other API classes that belong to their specific ODServer session.

Access to a single ODServer session must be done in a single-threaded fashion. That is, only one thread can access objects of a specified Content Manager OnDemand session at a time.

Since ODWEK version 7.1.2.7, all API functions are fully synchronized and thread safe. All operations within the API are synchronized. The only exception is the ODServer.cancel() method, which is discussed further in 6.6.3, "Implementing an application timeout by using the ODServer.cancel() method" on page 140.

Single-threaded access to a single Content Manager OnDemand session implies that when one thread accesses operations within an ODServer session, any other threads that try to access the same objects are blocked.

However, you cannot regard the synchronization that is built into the ODWEK Java APIs as the only level of synchronization when building Web applications. When each call to a servlet is wrapped in its own thread, ensure that each thread uses the right ODServer instance. In Web applications, you typically keep the ODServer object within the Web application server session data, which enables the threads to have access to the right ODServer objects that belong to the user session.

Guaranteeing a single ODServer instance per session does not eliminate the threading issue. A common issue in Web application is when users mistakenly submit multiple identical transactions. Examples include when a user clicks the Submit button of a form multiple times, when a user clicks the browser Stop button and resubmits a transaction, and when a user clicks a link several times. Each operation creates multiple transactions at a mid-tier level and a new thread. Each thread accesses the same ODServer instance, which violates the ODWEK Java API requirements of one thread at a time per ODServer instance. Subsequent access by multiple threads is blocked through the synchronization at the API level. To prevent possible loss of data or unexpected behavior, you must implement synchronization handling on the session object.

6.4.4 Synchronizing servlet code

By synchronizing on the servlet session object, you guarantee that multiple requests that come from the same browser are handled sequentially within your code. As shown in Example 6-7, synchronization on the session object is easy to implement and provides a safe environment in which to use the ODWEK Java APIs.

Example 6-7 Synchronization on a session object

```
public doPost(HttpServletRequest req, HttpServletResponse res) {
   HttpSession wwwsession;
   ODServer odserver;

   wwwsession= req.getSession(true);

   synchronized(wwwsession) {

      odserver = (ODServer)wwwsession.getAttribute("ODServer");
      ODFolder fld = odserver.openFolder("Statements");

      ...
   }

   ...
}
```

Note: By synchronizing on the session object or on the ODServer object, you can ensure that no other thread can perform operations on the same Content Manager OnDemand connection session.

Implementing synchronization at the application logic level is possible but not recommended. For example, you can synchronize on the specific application method or class that initiates the search in Content Manager OnDemand. The drawback of doing this is revealed by examining the thread flow of the Web application. You cannot be assured that, apart from (for example) the search operation, all other operations are synchronized. Otherwise, you limit the execution of the ODWEK Java API operations to a completely single-threaded model. Therefore, either carefully test the synchronization behavior of your application when using self-defined synchronization, or preferably, synchronize on the session object of the Web application.

Tracking down synchronization performance problems

You can run into performance problems when implementing too much synchronization or synchronizing in the wrong locations. One way to track down threading and synchronization problems within an existing application is to take a Java dump of the JVM in WebSphere (or any other Web application server that you are using) when the performance problem occurs. By examining the dump, you can determine the object or method that each thread is working on and which of the threads is waiting. Only threads that are in an *active* state and have com/ibm/edms/od/* classes in their stack trace are working actively on the ODWEK Java APIs.

Synchronization according to user sessions

In many Internet implementations, multiple concurrent sessions are established by using the same user name, which can lead to the idea of implementing per-user synchronization. Per-user synchronization is not necessary in Content Manager OnDemand.

Content Manager OnDemand supports the coexistence of multiple sessions that are all logged in by using the same user ID. All of the sessions can access the ODWEK Java APIs concurrently. The requirement is that only one thread operates on one ODServer session at a time. The user login that is used by the ODServer session is of no importance with respect to the multithreaded behavior of the ODWEK Java APIs. The only synchronization level that is required by the ODWEK Java APIs is at the session level, not at the user or login level.

6.5 Resource consumption control

Resource consumption, especially memory, is highly impacted by the proper initialization and termination of Content Manager OnDemand connections. In the following sections, we describe best practices to follow during connection initialization and termination for resource consumption control.

6.5.1 Connection initialization

ODWEK version 7.1.2.6 introduced the ODConfig class, which is used to configure an ODServer session without reading the arswww.ini file. In version 8.4, use of the ODConfig class is mandatory, because the previous ODServer constructor that worked without an instance of ODConfig has been deprecated.

Initialization steps

An ODWEK session can be initialized with the following actions:

1. An ODConfig object is instantiated and its configuration data is set.
2. An ODServer object is created, specifying the ODConfig object as a constructor parameter.
3. The ODServer.initialize() method is called to initialize a new Content Manager OnDemand client session, which then can be used to perform a logon to an OnDemand server.

The ODConfig object contains the configuration data and makes the configuration reusable when creating multiple sessions. ODConfig objects are configured though the constructor. An ODConfig object cannot be reconfigured, but its configuration can be queried by using multiple methods.

ODConfig parameters

The ODConfig class supports three constructors:

- `ODConfig()`

  The ODConfig() constructor initializes the object by using the default configuration values.

- `ODConfig(String afpViewer, String lineViewer, String metaViewer, long maxHits, String appletDir, String language, String tempDir, String traceDir, int traceLevel)`

  This ODConfig constructor is the default when specifying a custom configuration. All standard configuration settings that are available in ODWEK are represented by the parameters.

- `ODConfig(String afpViewer, String lineViewer, String metaViewer, long maxHits, String appletDir, String language, String tempDir, String traceDir, int traceLevel, java.util.Properties props)`

  This ODConfig constructor requires the same configuration options as the previous one, but includes a Properties object, which can hold additional properties. The Properties object is used for providing configuration data for the AFP2WEB and Xenos transformations.

Table 6-3 describes the available configuration parameters and the default values that are used when you create an ODConfig object without using any parameters.

Table 6-3 ODConfig configuration parameters

Parameter	Default value	Description
afpViewer	ODConstant.PLUGIN	Default conversion method or viewer for AFP data documents
lineViewer	ODConstant.APPLET	Default conversion method or viewer for line data documents
metaViewer	ODConstant.NATIVE	Default conversion method for metadata documents
maxHits	200	Maximum number of hits that can be returned
appletDir	/applets	Directory where the applets reside
language	ENU	Language used by ODWEK for messages
tempDir	java.io.tmpdir	Directory to store temporary files
traceDir	java.io.tmpdir	Directory to store the trace files in (if enabled)
traceLevel	0	Tracing level: 0: No logging enabled 1: Log only ERROR events 2: Log only ERROR and WARNING events 3: Log ERROR, WARNING, and INFO events 4: Log ALL events
props	N/A	Properties object for providing additional configuration data

For tempDir and traceDir, the default value is the path of the temporary directory that the system uses. This value is retrieved by using the `System.getProperty("java.io.tmpdir")` Java method.

The props parameter, which is an object of type Properties, is a container for configuration data that is needed for transformations. For each transformation (AFP2PDF, AFP2HTML, AFP2XML, and Xenos), the directory of the transformation binaries and the location of the INI configuration file must be set in the props object.

Special note about trace files and language

The trace directory, traceDir, which is used for trace files, can grow to a significant size, depending on the traceLevel. ODWEK creates a new trace file each time the Web container is restarted and renames the existing file by using a date-time scheme. The trace files are not deleted automatically, and therefore, must be deleted manually.

The language parameter configures the language in which ODWEK produces messages. The only type of messages that you receive from ODWEK classes are exceptions. The language setting is irrelevant unless you plan to display the exception messages to users. If this is the case, you must set the language parameter to the language in which your application is running. See Chapter 7, “Globalization” on page 143, for a list of supported languages. If you do not want to expose the exception message to users, you can create your own messages.

Language parameter: The language parameter is a global setting that applies to the whole JVM. The first call to ODConfig sets the language for all applications running in this JVM. The language parameter cannot be changed after the first call. All subsequent changes to language are ignored.

The easiest way to initialize an ODWEK session is to use the default configuration. Example 6-8 shows the sample code to initialize an ODWEK Java API session.

Example 6-8 ODWEK session initialization

```
try{
   ODConfig cfg = new ODConfig();
   ODServer odserver = new ODServer(cfg);
   odserver.initialize("AppletCallbackServlet");
   odserver.logon("9.155.41.12", "user01", "secret");
   ...
} catch(ODException e){
   ...
}
```

6.5.2 Logging off and terminating a client connection

The ODServer class is reusable with regard to user logins. After an ODServer.iniialize() call, you can use the ODServer.logon() and ODServer.logoff() methods multiple times on the same ODServer class instance. You can have your application log off a user and log on another user by using the same ODServer object.

The ODServer.logon() method creates a logon token. The ODServer.logoff() method removes the logon token.

After the final ODServer.logoff() method is called, the ODServer.terminate() method must be called. The ODServer.terminate() method informs the application that the client session is finished and decrements the number of clients-in-use counter. When the counter reaches zero, then all shared resources are deleted.

Attention: Failing to call the ODServer.terminate() method causes the shared resources not to be released.

Consider the following points when logging off and terminating a client connection:

- A session that is terminated by an ODServer.terminate() call before an ODServer.logoff() call becomes unusable, but the logon token still persists on the system for a while until it times out.
- If an ODServer class instance is not used anymore, for example, because of an exception in the application, an explicit logoff, a close on the client interface, or a timeout of the Web application, ensure that the ODServer.logoff() and ODServer.terminate() methods are called. The best practice is to call the ODServer.logoff() method (if an active logon still exists) and then call the ODServer.terminate() method.
- Ensure that your exception handling code contains functionality to correctly terminate a Content Manager OnDemand session. When handling a critical application error, which can force a user to actively logon to your application again, make sure that you end the ODServer session by using the ODServer.logoff() method (if applicable) and calling the ODServer.terminate() method.
- When handling exceptions within your login or initialization code, see if ODServer has been initialized before calling the terminate() method because the exception might be thrown during the ODServer.initialize() call. To check whether the ODServer is initialized, use the ODServer.isInitialized() method.

6.5.3 Allocation and release of resources and sessions

Three main types of data are stored in memory when using the ODWEK Java APIs:

- Objects retrieved from the OnDemand server

  Each time a document is retrieved, it is stored temporarily in memory while it is downloaded to the ODWEK Java APIs. If your application design permits, you can use less memory by downloading documents directly to disk. See Chapter 9, “Document retrieval” on page 179, for more information.
- Shared AFP resources that are retrieved along with the AFP data stream

  AFP resources are loaded into memory by the ODWEK Java APIs. All further requests for AFP documents that use the same resource are served by using the resource data that is located in memory. The resource data is not freed until all active client connections are terminated.

- Hitlist

  Only one active hitlist is cached in memory for each session (ODServer instance). Each time a search is performed, the resulting hitlist is cached in memory and is overwritten by the next search.

In addition to these classes, Java class instances, logon credentials, handles to open folders, and other minor data structures are reserved during the lifetime of their objects.

In addition to the objects in memory during retrieval and the hitlist creation, the AFP resources are the main consumers of memory. They are loaded into memory the first time that a user requests an AFP document, but they are not released as long as there is an open session.

Every time you initialize an ODWEK session, by using the ODServer.initialize() method, an internal client counter is *incremented*. Each time you call an ODServer.terminate() method to terminate the session, the internal counter is *decremented*. This way, the application keeps track of the number of sessions it must maintain. If the internal client counter reaches zero, ODWEK is ensured that no sessions want to access the shared resources again. Therefore, it releases all shared resources and the used memory.

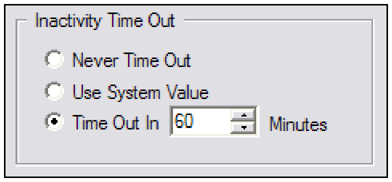
Important: If you do not end any ODServer session properly (with an ODServer.terminate() method), then the client counter cannot reach zero, and the shared resources are never released. The shared resources stay in memory until you restart the application server. It is critical that you properly end each session by using ODServer.terminate() method.

6.6 Timeout

In this section, we discuss the inactivity timeout, which is defined to Content Manager OnDemand and can be used to interrupt sessions. We also discuss methods by which you can create your own timeout.

6.6.1 Inactivity timeout

The *inactivity timeout* keeps track of the amount of time that has expired since the last application call to the OnDemand server. You can set the inactivity timeout in the inactivity timeout configuration dialog box (Figure 6-5) on the System Parameters tab of the administration client. You can set the inactivity timeout for individual users. You can set it to never time out, use the system value, or time out within a defined length of time.


Inactivity Time Out

- ( ) Never Time Out
- ( ) Use System Value
- [✓ Time Out In] 60 Minutes

Figure 6-5 Configuring the inactivity time out in the administration client

The inactivity timeout does not force a connection termination or logoff. The timeout is internally maintained by the API code at client level and is an indication that the logon session for the user has expired due to inactivity.

When an inactivity timeout occurs, an exception of type ODException is thrown. The connection session itself is not affected by this. You are not required to call the ODServer.terminate() and ODServer.initialize() methods. However, because the user's logon session has expired, you are required to re-login the user. To accomplish this, you must call the ODServer.logoff() method first to properly terminate the user's previous logon session. Then your application can call the ODServer.logon() method to login the user again.

Resetting the inactivity timeout counter: You can reset the inactivity timeout counter through any API operation that communicates with the OnDemand server.

The inactivity timeout can be used to help ensure your application knows when to properly terminate the API objects so that it does not cause memory issues. As mentioned in 6.1, "Connection pooling overview" on page 112, it is important to call the ODServer.terminate() method after a session is not required anymore. Issuing the ODServer.terminate() method decrements an internal Content Manager OnDemand counter, allowing the ODWEK Java APIs to recognize when there are no more required connections. That is when the counter is at zero. When this occurs, the ODWEK Java APIs release any unneeded shared resources from memory.

The inactivity timeout allows your exception handling code to check if you still need the connection session (based on how your client interface works). If you do not need the connection session, call the ODServer.logoff() and ODServer.terminate() methods to finalize and terminate the unused connection.

When implementing connection pooling, consider how the inactivity timeout is used:

- Setting the inactivity timeout period to *Never Time Out* allows the Content Manager OnDemand connections to remain active until the OnDemand server is shut down. In this case, make sure that no ODServer object is lost and that it is terminated when appropriate.
- When enabling an inactivity timeout even on the pooled connections, you must write your own code to keep the connection alive. You are required to handle the session termination in the exception-handling code.
- Starting with ODWEK version 8.4.0.2, the ODServer.keepServerAlive() method is added to reset the timeout counter. You can use this method to prevent a user session from being timed out.

6.6.2 Other timeouts

Other than the inactivity timeout, which is a user- and session-based timeout, Content Manager OnDemand does not incorporate any other timeout concepts. Specifically, there is no operation-based timeout concept for breaking operations that last too long. Because of this, from the Content Manager OnDemand perspective, any operation in ODWEK theoretically executes until the operation is completed or stops when an error is returned.

Depending on how you design your application, a timeout can be enforced by the Web application server. For example, you can configure a session timeout to cancel the execution of a Web application after a defined time. In this case, check to see what happens within the Java application at the time that the Web application server enforces the timeout:

- If a Java exception is thrown, make sure that your application is informed that your request is no longer needed.
- If the timeout occurs within a search or retrieve operation, call the ODServer.cancel() method. See 6.6.3, “Implementing an application timeout by using the ODServer.cancel() method” on page 140.
- If the timeout renders the current session invalid or no longer in use, terminate it correctly. See 6.5.2, “Logging off and terminating a client connection” on page 135.

6.6.3 Implementing an application timeout by using the ODServer.cancel() method

By using the ODServer.cancel() method, you can implement an application timeout on operations, for example, search and retrieve, that might run for a long period of time. The ODServer.cancel() method can cancel both the search and retrieve methods. Calling the ODServer.cancel() method during any other operation has no effect.

Calling the ODServer.cancel() method has the following effects:

- Search (ODFolder.search)

  Calling a cancel during a search causes the current search to be aborted. A partial hitlist that contains all hits that are found up to the point is returned as the search result.

- Retrieve (ODFolder.retrieve, ODHit.getDocument, ODHit.getResources, ODHit.retrieveSegment)

  Calling a cancel during a retrieve process causes the retrieval process to be aborted and an ODException to be thrown.

  In a Web application, when retrieving large documents that cause extended wait times (for example over 5 or 10 seconds), consider using one of the following methods:

  - The ODHitProperties.getLength() method to obtain the length of the document (in bytes) before retrieving it
  - Informing the user of a possible long retrieval process (if the number of bytes is large)

    This method helps to improve user experience and prevents the user from canceling the retrieval process in the middle of the operation.

If you are developing custom client applications, consider offering a Cancel button or link in your application to enable users to cancel these two types of operations.

If you are developing non-interactive applications, consider wrapping search or retrieve calls with timeout functions. By doing so, the application calls the cancel method if the search or retrieve operation does not return within a defined period of time. You can accomplish this wrapping by using an extra thread that calls the ODServer.cancel() method when the operation does not return with a specified time frame.

6.6.4 Recommended timeout implementation for a Web application

In Web applications, users do not synchronously communicate with the application. Instead, all interactions are done by requesting content through the use of a Web browser. Users tend to ignore this fact and act as though they are using a standard application, which may lead to problematic situations. We describe one such situation in the following section.

Situation: Perceived non-responsive application

If a Web application performs a search that takes a long time and the user clicks the Stop button on the browser window, the user expects the search to be cancelled. In reality, the Stop button causes the *browser* to stop retrieving data. Because there is no direct interaction with the application, the search in Content Manager OnDemand continues.

If the user wants to perform another search, a new search request is issued for the same OnDemand session. Because the ODWEK Java APIs are synchronized on the session object, only one search operation can be performed in one session. Therefore, when the user clicks the Stop button, the first search still carries on for the OnDemand session. The new search (or any other operation) that is issued by the same user is blocked until the first search is finished. From the user's perspective, the result is a non-responsive application.

To avoid this behavior, your application must be able to detect a running search or retrieve operations. For every call made to a session, check if a search or retrieve operation is currently in progress. If such an operation is in progress, consider canceling it before executing the next search or retrieve operation.

A solution: Use an operation-ongoing flag

Detecting an ongoing search or retrieval operation can be implemented by introducing a flag in the session object. The flag is set before the search or retrieve operation starts and is reset when the operation is completed. All search or retrieve requests check the flag before making calls to the API. If the flag is set, then the user has either stopped the browser and issued a new request or the user is performing a parallel action.

Multiple tab functionality: Many browsers offer the ability of opening multiple tabs. Users can use this functionality to issue parallel requests in the same session. For example, if a user is viewing a hitlist and wants to retrieve two different documents in two new tabs, then two retrieve calls are made in parallel.

Cancelling the search or retrieve operation that is currently in progress in order to start a new request might lead to strange effects from the user's perspective. As described in the previous example, opening two documents at the same time causes the first document to return an error and the second document to be retrieved.

You can solve this problem by using an operation-ongoing flag. When the first search is performed, the flag is set to be *true*. If another search is requested and the operation-ongoing flag is already set to true, then you can display a window that informs the user of the ongoing operation. Instead of simply canceling the existing running operation and starting a new one, the window should offer the user the ability to cancel the existing running operation, in favor of the new request, or cancel the new request.

If the page contains a self-reloading mechanism, which reloads the page every second and checks every time if a flag is still set, then you can also satisfy users who work with multiple tabs. In the example mentioned previously, now the first document retrieval runs, and the second tab shows the information page until the first retrieval is finished. Afterward, Content Manager OnDemand retrieves the second document automatically.

7

Globalization

In this chapter, we discuss globalization in association with IBM Content Manager OnDemand (OnDemand) and OnDemand Web Enablement Kit (ODWEK). Many of the Content Manager OnDemand implementations do not run in a homogenous environment. Multiple clients that use the same application may be running with different languages, locale, and code page settings. Even with a homogenous environment, the Content Manager OnDemand implementation might use languages and code page settings that are different than standard default.

To ensure successful implementation of the Content Manager OnDemand system, you must understand and take into consideration the usage of different encoding schemes and conversions.

This chapter covers the following topics:

- Globalization overview
- Content Manager OnDemand character conversion architecture
- Code page conversion in ODWEK
- The ICU conversion library
- Using Unicode as the database code page
- ODWEK language configuration
- Integrating custom code pages
- Globalizing applications by using ICU

7.1 Globalization overview

In computing, *globalization* is the provision of a single software solution that has multicultural support and a user interface and documentation that is available in one or more languages. *Multicultural support*, in this context, is the ability of a single software solution to be translatable and to support the cultural conventions of multiple languages and geographic regions. Cultural conventions include the use of various writing systems, sort orders, different formats for date, time, numbers, and currency, different keyboard layouts, and others. It is similar to the term *National Language Support*. In this book, we focus our attention on multicultural support.

Content Manager OnDemand is designed for the global market for worldwide distribution. Content Manager OnDemand can be configured to work in different locales to support various languages. The OnDemand server runs in *Unicode* so that client interfaces can be displayed in various languages.

ODWEK uses Unicode Transformation Format 8 (UTF-8), which is the world-wide Web standard for its internal encoding that enables it to have multicultural support of different languages. The languages that are supported in the ODWEK client interface are bundled as resources in the client application. Therefore, they are bound to the client installation. Multiple clients running in different languages can access the same OnDemand server concurrently. The Content Manager OnDemand system handles various languages and character sets by using *code pages* and translation facilities that translate data from one code page into another.

Code page: A *code page* is a defined character map. Content Manager OnDemand uses the Unicode character model for code pages.

Unicode is an international industry standard that enables computers to represent and manipulate text characters for most written languages. Unicode consists of a set of more than 100,000 characters and text elements, each of which is defined with a unique code. Unicode is closely aligned with international standard ISO/IEC 10646, which is also known as the Universal Character Set (UCS).

Each code page represents a mapping of the Unicode characters for a specified language and locale, ranging from one byte to multiple bytes per character (single byte character set (SBCS), double byte character set (DBCS), and multiple byte character set (MBCS)).

For example, ISO 8859-1 is one of the most common code pages that is used to represent western European languages. In ISO 8859-1, the first 127 characters are the standard ASCII characters. They are followed in positions 128 through 256 by many of the signs and characters that are used in western European languages.

Different countries and languages use different code pages. Custom font and character sets, such as those used in Advanced Function Presentation (AFP)-applications, might also have their own code pages.

If a Content Manager OnDemand system uses a single code page, then no code page translation is required. However, consider an example where line data spools are loaded from an external system that runs an Extended Binary Coded Decimal Interchange Code (EBCDIC) 500 code page and the Windows clients use Windows-1252 code page. In this case, without code page conversion, users cannot read the displayed data on their client application because the data stream bytes are mapped with different values in different code pages.

A portion of the Windows-1252 code page is shown in Figure 7-1, and portion of the EBCDIC 500 code page is shown in Figure 7-2 on page 146. For both code pages, each character is shown with its Unicode equivalent underneath and its decimal code at the bottom.

For example, the small letter “a” has a Unicode value of 97 in Windows-1252 code page. However, the same value 97 on the EBCDIC 500 code page represents the forward slash character (/). The small letter “a” on the EBCDIC 500 code page is represented by 129. Without code page conversion, the data that is loaded from the external system using EBCDIC 500 code page is not correctly represented by the Windows client application, where the Windows client application uses the Windows-1252 code page.

	–0	–1	–2	–3	–4	–5	–6	–7	–8	–9	–A	–B	–C	–D	–E	–F
6–	\` 60 ***96***	a 61 ***97***	b 62 ***98***	c 63 ***99***	d 64 ***100***	e 65 ***101***	f 66 ***102***	g 67 ***103***	h 68 ***104***	i 69 ***105***	j 006A ***106***	k 006B ***107***	l 006C ***108***	m 006D ***109***	n 006E ***110***	o 006F ***111***
7–	p 70 ***112***	q 71 ***113***	r 72 ***114***	s 73 ***115***	t 74 ***116***	u 75 ***117***	v 76 ***118***	w 77 ***119***	x 78 ***120***	y 79 ***121***	z 007A ***122***	{ 007B ***123***	| 007C ***124***	} 007D ***125***	~ 007E ***126***	DEL 007F ***127***
8–	€ 20AC ***128***	***129***	‚ 201A ***130***	ƒ 192 ***131***	„ 201E ***132***	… 2026 ***133***	† 2020 ***134***	‡ 2021 ***135***	ˆ 02C6 ***136***	‰ 2030 ***137***	Š 160 ***138***	‹ 2039 ***139***	Œ 152 ***140***	***141***	Ž 017D ***142***	***143***

Figure 7-1 Partial Windows-1252 code page

	–0	–1	–2	–3	–4	–5	–6	–7	–8	–9	–A	–B	–C	–D	–E	–F
6–	- 002D ***96***	/ 002F ***97***	***98***	***99***	***100***	***101***	***102***	***103***	***104***	***105***	¦ 00A6 ***106***	, 002C ***107***	% 25 ***108***	_ 005F ***109***	> 003E ***110***	? 003F ***111***
7–	***112***	***113***	***114***	***115***	***116***	***117***	***118***	***119***	***120***	` 60 ***121***	: 003A ***122***	# 23 ***123***	@ 40 ***124***	' 27 ***125***	= 003D ***126***	" 22 ***127***
8–	***128***	a 61 ***129***	b 62 ***130***	c 63 ***131***	d 64 ***132***	e 65 ***133***	f 66 ***134***	g 67 ***135***	h 68 ***136***	i 69 ***137***	***138***	***139***	***140***	***141***	***142***	± 00B1 ***143***

Figure 7-2 Partial EBCDIC code page

7.2 Content Manager OnDemand character conversion architecture

From a code-page perspective, the Content Manager OnDemand system contains two types of data:

- Reports and documents

  All report and document data types, such as line data, AFP, and images, are included. This data is stored into Content Manager OnDemand either through a load process by using ARSLOAD (manual or automated) or by using the ODWEK Java APIs. The data is stored on the Object Server.

- Indexes and annotations

  The indexes are created during the load process and enable users to search reports and documents in the Content Manager OnDemand system. The annotations are added by system users and include either additional information or events that are related to the documents. Both the index and annotation data are stored in the database that is used by the Content Manager OnDemand Library Server.

Figure 7-3 on page 147 shows the Content Manager OnDemand and ODWEK data encoding architecture. It includes areas in which different code pages are used for these two types of data and where the code page conversions occur. The diagram illustrates the Content Manager OnDemand instance, the ODWEK instance, the Web browser, and the data encoding between the different components. The cogged (toothed) wheels represent the points within the system where the data conversions occur. As a general rule, all data interfaces to the server are in UTF-16 data, except for document data, which is transferred in its native code page.

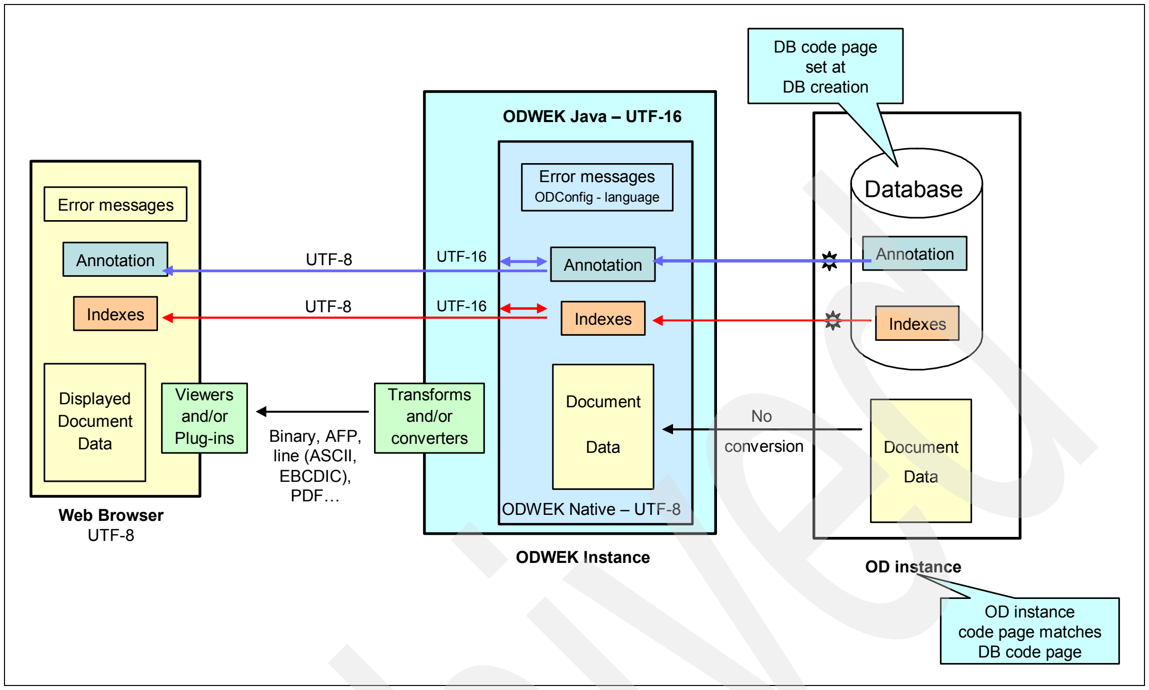


Figure 7-3 The Content Manager OnDemand and ODWEK data encoding architecture

More information: To fully understand the concept of how Content Manager OnDemand deals with data conversion, read Appendix J, “National Language Support” in the *DB2 Content Manager OnDemand for Multiplatforms Ver 8.4: Installation and Configuration Guide*, SC18-9232.

7.2.1 Index data and annotations conversions

At the load time, the index data is stored in the Content Manager OnDemand database tables, by using the database code page, on the Library Server. The arsload process produces the indexes and can run on a different machine by using a different code page. Thus, the indexes that are generated by the arsload process might need conversion before they are stored in the Content Manager OnDemand database. The document data is always stored in the Content Manager OnDemand archive in its native form. This means that there is no data conversion for document data.

At retrieval time, the indexes and annotation data are converted to UTF-16 for transport. The indexes and annotations are then converted to the client code page by the client.

The database code page is set at database creation time and cannot be changed. When creating an Content Manager OnDemand instance, its code page must match the database code page. Therefore, no data conversion is required between these two components.

Note: The OnDemand server runs in the same code page as the Content Manager OnDemand database.

7.2.2 UTF conversion

UTF-16 is a variable-length UTF, which encodes each Unicode character into two or four bytes. It can be used to represent almost any character from any code page.

TCP/IP traffic that is composed of indexes, annotations, and control data (such as user-login, folders, and other internal data) and is transferred between a server and client (regardless of client type, ODWEK, or Windows) is always transferred in UTF-16. All index and annotation data are converted from the native database and server code page to UTF-16 for data transmission. Later the UTF-16 data is converted into other code pages by the client, for example, to UTF-8 for ODWEK or to the Windows locale for a Windows client.

The UTF-16 conversion is required since the server does not know which code page a client is using. Therefore, the server cannot send the data in the client's code page directly. Additionally, the client might not know the code page that the Content Manager OnDemand instance and database are using. By standardizing the transmission code page, both the client and server can be more efficient in handling conversions.

To the OnDemand server, ODWEK is just another client. Since Web browsers transmit and display UTF-8 data, ODWEK is designed to work completely in UTF-8. This allows ODWEK and the Web browsers to directly exchange data by using UTF-8 without undergoing any further code page translations. Data that is exchanged with the OnDemand server is converted automatically from UTF-8 to UTF-16 and vice versa.

The Java layer of the ODWEK environment uses UTF-16, because UTF-16 is the only native Java code page. The conversion between Java's UTF-16 code page and ODWEK's native UTF-8 code page is done automatically within the ODWEK Java APIs.

7.2.3 Document data conversions

Document data is generally stored untouched in its native format. When the data is loaded by using the Content Manager OnDemand load mechanisms (arsload, Store API, Monitors, Indexer), no data conversion is applied. The data is stored as is, in its native code page and format.

The only moment in time in which conversions on data come into play is at the presentation level. If required, conversions and handling of custom font mappings (for AFP) are done on the client. A line data document that is archived on an EBCDIC computer cannot be displayed on a Windows computer without either a code page conversion or a viewer that is capable of displaying the EBCDIC code page on the client.

Depending on the data and type of the client, different processes occur on the client computer system. The standard Windows client incorporates a line data viewer component and viewing capabilities for AFP documents. Both modules internally map the code pages and display the characters in the way they should be displayed. For binary document data, and data that is in a format or code page unknown to the viewers, the native document data is passed to the associated application for further processing.

7.3 Code page conversion in ODWEK

ODWEK code page conversion behavior is somewhat different from that of the standard Content Manager OnDemand Windows client. ODWEK is a mid-tier system. It contains an additional presentation layer, usually a browser, but it can also be a stand-alone Java application by using the ODWEK Java APIs.

7.3.1 API conversions

ODWEK natively runs in UTF-8. Therefore, all indexes and annotation data that is sent from the OnDemand server must be converted from UTF-16 to UTF-8.

When implementing a Java application by using the ODWEK Java APIs, you must be aware of the code conversions that are taking place. Java internally operates in UTF-16. Data that is returned from the ODWEK Java API functions are automatically converted from UTF-8 to UTF-16 by Java. When you pass strings from Java to the ODWEK methods, you do not need to perform any additional tasks. The UTF-16 data is converted by the native ODWEK code to UTF-8.

See Figure 7-3 on page 147 to see the data code pages that are exchanged between the ODWEK code and a browser.

7.3.2 Browser conversions

When sending data to the browser, ODWEK does not do any other conversions on indexes or annotations. Therefore, a browser that displays index or annotation data that is received through the ODWEK Java APIs must be capable of handling UTF-8 Unicode data.

If you are delivering index data to any external applications by using your ODWEK-based Java application, make sure that those applications can handle Unicode data. Otherwise, you must convert the data manually. The same implications apply if you save any index or annotation data to a file. If you do not do any explicit conversion, the data is written as a Unicode data stream.

Web browsers usually are capable of displaying and sending UTF-8 data. Therefore, this does not present itself as an issue when implementing Web applications.

7.3.3 Document data conversions

For the actual document data, conversion can be handled in the following different ways:

- When you request raw native document data, ODWEK returns the data in its unaltered form, in the same code page in which it was archived.
- When you request that the line data to be displayed as an applet, ODWEK internally sends the UTF-8 ASCII data to the applet, and you receive the standard HTML containing applet invocation code.
- When you request an ASCII conversion, ODWEK returns a UTF-8 ASCII converted representation of the original AFP or line data document.
- For most other document types, ODWEK behaves in the same manner as the Content Manager OnDemand Windows client. It passes the data back in its native format.

For more details about document data conversions in ODWEK and how you can request data in different formats, see Chapter 10, “Applets, plug-ins, and transforms” on page 197.

7.4 The ICU conversion library

Content Manager OnDemand performs conversion or mapping from different code pages to other code pages by using a standard component called *International Components for Unicode (ICU)*. ICU is an open source project that was developed by IBM and other companies. It is a library that is available for Java and C and is used for multicultural support. ICU provides services such as character conversion between different code pages, language-sensitive collation, and locale and resource management support.

In earlier versions of Content Manager OnDemand, the ICONV library conversion engine was used. Content Manager OnDemand was updated to ICU in version 7.1.2.1 (AFP viewer components in Version 7.1.2.5). ICONV is a character encoding library that is primarily distributed on UNIX® environments as part of the GNU C library in most Linux distributions.

ICU is used by Content Manager OnDemand at different locations for code page conversion and text operations. Figure 7-4 shows the components within the Content Manager OnDemand architecture that the ICU library uses to convert data.

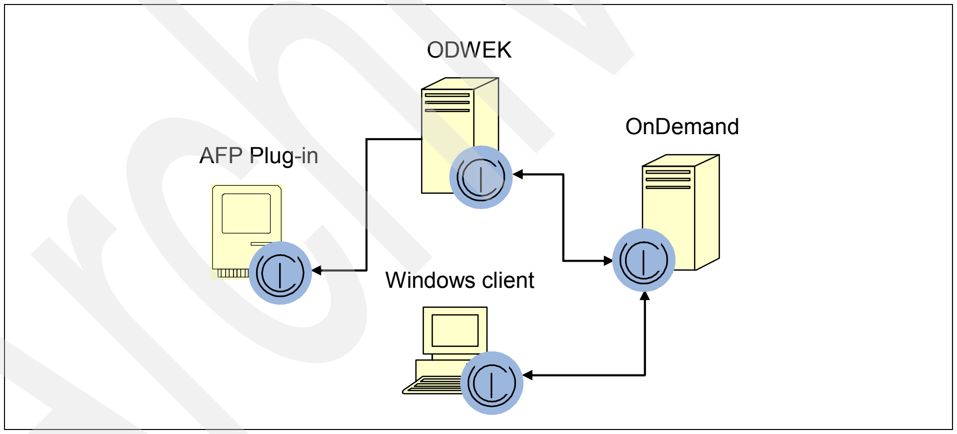


Figure 7-4 Content Manager OnDemand components that use the ICU library

Each component uses ICU for different use cases, but all do code page conversions, which enables communication with other parts of the Content Manager OnDemand infrastructure:

- The OnDemand server uses ICU for index and annotation conversions. All TCP/IP traffic is in UTF-16, but the index and annotation data are stored in the database code page format. Therefore, ICU is used for data conversion between the database instance code page and UTF-16.

- ODWEK uses ICU to convert the index and annotation TCP/IP data, which is in UTF-16, to the internal data format used in ODWEK, which is UTF-8. Additionally, if you are using line data and the line data Java applet, then the actual data is converted from its native code page to UTF-8, which is used by the applet. All other documents are passed through without conversions.
- The standard Content Manager OnDemand Windows client uses its ICU facility to convert AFP and line data to the local Windows code page for viewing by using internal viewers.
- The AFP plug-in, which works similar to the AFP viewer that is integrated within the Content Manager OnDemand Windows client, has its own ICU library for converting AFP data to the local Windows code page.

7.5 Using Unicode as the database code page

Generally, when you develop OnDemand applications by using ODWEK, your database is already created. The code page of the database is set, and you cannot change it. Therefore, the discussion of using Unicode as the database code page or not using it is not relevant at this point. However, since this chapter covers globalization, we include the discussion for your reference. This information might be useful for future implementations.

Tip: Using Unicode as the database code page can help to solve some of the issues that you might experience when dealing with multiple code pages and languages.

In most Content Manager OnDemand implementations, the OnDemand server and the Content Manager OnDemand database are configured to use the operating system locale. The operating system locale is either a single-byte or a double-byte code page (for example, a Windows-1252 code page). Since modern databases are capable of working directly in Unicode, the ability to configure the OnDemand server instance to use Unicode has been added to the system.

You can choose to run an OnDemand server instance by using Unicode encoding UTF-8 or UTF-16 as long as you ensure that the database that you use for the Content Manager OnDemand instance uses the same Unicode encoding.

When Content Manager OnDemand and database Unicode encoding are needed

In most scenarios, it is not necessary to use Unicode for the server and database encoding. If you are working in an environment in which data from different code pages must be handled, Unicode encoding might help to store the data correctly.

For example, an OnDemand server instance uses a Japanese code page because the users and applications that are working on this instance are in Japan. Additionally, you must archive European documents that contain the euro sign character (€) in their indexes. The problem is that the Japanese code page does not contain the € character. Without using Unicode, the solution is to set up two different OnDemand server instances, one running in a Japanese code page and the other one running in a European code page. However, in this case when your Japanese users want to add annotations (in Japanese) to the European documents, you might have problems. In such a scenario, use Unicode.

Unicode and application group field length

When a database uses Unicode instead of a single-byte or multiple byte code page, be aware of the possible impact on the length of the application group field.

Unicode encodings, such as UTF-8 or UTF-16, represent characters as a variable number of bytes. For example, UTF-8 uses one to four bytes to store a character. A character within the standard ASCII range of characters, such as the Latin letters (a-z and A-Z), is stored by using a single byte. Umlauts (for example "ä") are stored by using two bytes (in this case 0xC2 0xAE). The € character is stored by using three bytes (0xE2 0x82 0xAC). Some Asian characters are stored by using four bytes.

The different byte-length for each character can lead to issues concerning the maximum length of the database application group fields in Content Manager OnDemand. Each application group field in Content Manager OnDemand corresponds to a field in a database table. When creating an application group string field, you specify the maximum character length of this field. Internally at the database level, the field length is not enforced by the number of characters. Instead, it is calculated based on the number of bytes.

For example, a string field with a maximum length of 10 in a Unicode database can hold 10 standard ASCII characters. In the worst case, it can hold only two 4 byte characters. When loading index data into Content Manager OnDemand, all characters that cannot be stored because the field length is exceeded are stripped off.

Therefore, if an OnDemand server instance and database are set up to work in Unicode, the string field length must be set to a sufficient length.

7.6 ODWEK language configuration

The ODWEK Java APIs do not need to be configured with code pages or character sets. Internally, ODWEK uses Unicode UTF-8 encoding. The surrounding Java API code uses Unicode UTF-16. All TCP/IP communication between the OnDemand server and ODWEK is in UTF-16. The ICU conversion engine automatically handles the data conversion.

Important: If you work with other pieces of ODWEK, especially the CGI module, you have to specify the code page of the OnDemand server and Content Manager OnDemand database in the arswww.ini file. This setting is not required for the ODWEK Java APIs.

In this section, we discuss how to set language for ODWEK output and what you need to pay attention to when dealing with multiple languages.

Supported languages

The multicultural support that is provided by ODWEK is the same as the multicultural support that is provided by other components of Content Manager OnDemand. You can specify a language in which all messages are displayed. Table 7-1 shows the list of available languages that can be used with ODWEK.

Table 7-1 Languages supported by ODWEK

Language code	Region or country	ISO and Windows code page
ARA	Egypt	ISO8859-6 / 1256
CHS	China	IBM-eucCN / eucCN / GBK
CHT	Taiwan	IBM-eucTW / eucTW / Big5
CZE	Czech	ISO8859–2 / 1250
DAN	Denmark	ISO8859-1 / 1252
DEU	Germany	ISO8859–1 / 1252
ENU	English (US)	ISO8859–1 / 1252
ESP™	Spain	ISO8859–1 / 1252
FIN	Finland	ISO8859–1 / 1252
FRA	France	ISO8859–1 / 1252
FRC	Canada	ISO8859–1 / 1252

Language code	Region or country	ISO and Windows code page
HRV	Croatian	ISO8859–2 / 1250
HUN	Hungarian	ISO8859–2 / 1250
ITA	Italy	ISO8859–1 / 1252
JPN	Japan	IBM-eucJP / eucJP / IBM-943
KOR	Korea	IBM-eucKR / eucKR / 1363
NLD	Netherlands	ISO8859–1 / 1252
NOR	Norway	ISO8859–1 / 1252
PLK	Polish	ISO8859–2 / 1250
PTB	Portugal / Brazil	ISO8859–1 / 1252
RUS	Russia	1251
SKY	Slovakian	ISO8859–2 / 1250
SLO	Slovenian	ISO8859–2 / 1250
SVE	Sweden	ISO8859–1 / 1252

The value that is used to configure ODWEK is the language code (the first column in Table 7-1). The third column in Table 7-1 provides additional information about the code page that is internally used when data is represented in the selected language. For the double-byte languages (CHS, CHT, JPN, and KOR), the code pages are used by AIX, Solaris™, and Windows. Because this is handled internally, you do not need to configure the code page information, according to the ODWEK language settings.

System i installations: Due to the architecture of System i software, the language locales that are supported might differ from Table 7-1. See the "Locales," "Specifying the arswww.ini file," "[CONFIGURATION]", and "LANGUAGE" sections in *IBM Content Manager OnDemand for i5/OS® Common Server ODWEK Installation and Configuration Guide*, SC27-1163.

Configuring the ODWEK language

The language that is used by ODWEK is configured like all other configuration settings by using the ODConfig object. The constructor of the ODConfig class requires String parameter language. You must specify one of the language codes listed in column one of Table 7-1 on page 154. If you do not specify configuration values (by using the parameterless constructor), the default locale ENU (English/USA) is used.

Special attention for multi-language applications

You cannot use different languages in your application by initializing different sessions with different language locales. When a Content Manager OnDemand session is initialized by using an ODConfig object for the first time, the language is set for the entire Web application. For subsequent session initializations, the language setting is ignored.

If you must have multiple ODWEK message languages, deploy your application as multiple Web applications. However, before doing deploying the application, consider whether the language used by ODWEK for messages is really of interest to your users. Your users might never see any of the messages if you do not use ODWEK applets or expose other ODWEK material directly to the users.

7.7 Integrating custom code pages

If you use custom code pages in your AFP or line data document, or if you modified a code page because your business application or your printing system requires it, Content Manager OnDemand does not know the correct code page to use. The same situation occurs if you use a code page that is not known to Content Manager OnDemand.

In such cases, configure the ICU conversion engines in your Content Manager OnDemand system to correctly handle your custom code page.

7.7.1 Locations that require configuration

You must update the code page settings for the following client applications:

- Content Manager OnDemand Windows client
- The AFP plug-in
- ODWEK
- The stand-alone AFP viewer

Content Manager OnDemand Windows client

The ICU engine of the Windows client is responsible for rendering the AFP and line data documents that are viewed by the client. If users are running the Windows client, configure your custom code pages there.

AFP workbench viewer: The Content Manager OnDemand Windows client uses the AFP workbench viewer, which is installed automatically during the client setup. However, as the components of the workbench viewer are directly integrated into the client, the ICU installation of the client is used to render AFP documents. You do not have to configure the AFP workbench viewer installation if you view custom code pages in the Content Manager OnDemand Windows client.

AFP plug-in

If your Web application is running in an environment where the AFP plug-in is deployed, then you must configure custom code pages.

ODWEK

ODWEK uses the ICU engine for two data conversions.

The first data conversion is for UTF-16 data (indexes and annotations as they are transferred by using the TCP/IP connection to the server) to UTF-8, which is used by the internal native part of ODWEK.

The second conversion is for line data documents from their native code page in which they are stored in the server to UTF-8 when they are passed to ODWEK. This conversion is required when an application uses the line data Java applet. Therefore, you must configure the custom code page data in the ODWEK ICU if you want to use the line data Java applet.

Stand-alone AFP viewer

If your Web application works in an environment where the AFP plug-in is not used or is only partially used and the stand-alone AFP viewer application is used in all other cases, then configure your custom AFP code pages in the viewer application. For more information about configuring custom code page mappings, continue with the following sections.

7.7.2 ICU and ICONV

In version 7.1.2.1, Content Manager OnDemand upgraded its internal multicultural support library from using the ICONV character set conversion engine to the ICU library 3.0. However, the AFP workbench viewer, which is used by the Content Manager OnDemand windows client, did not upgraded until version 7.1.2.5. In Content Manager OnDemand version 8.4.0.0, the ICU library was updated throughout the product from ICU version 3.0 to ICU version 3.6.

If you already configured the ICONV engine for your custom code page data, you must reconfigure it for ICU, because the ICONV code page integrations works differently from ICU. If you already created custom code page data for ICU version 3.0, you must recompile the data when you migrate to Content Manager OnDemand version 8.4, because the ICU data files of ICU version 3.6 are not compatible with ICU version 3.0.

7.7.3 Customizing code page mappings for ICU

To customize code page mappings for Content Manager OnDemand Client Version 8.4 or later by using ICU version 3.6 data formats, see the “Customizing code page mappings for the OnDemand Client version 8.4.0.0 and later” technote at the following address:

http://www.ibm.com/support/docview.wss?rs=129&context=SSEPCD&dc=DB520&dc=DB560&uid=swg21290484&loc=en_US&cs=UTF-8&lang=en&rss=ct129db2

You can also see the Content Manager OnDemand for Multiplatforms Support page at the following address:

http://www.ibm.com/software/data/ondemand/mp/support.html

On this page, in the search field, type the following search string:

```
Customizing code page mappings for the OnDemand Client
```

Several articles are returned from the search. Select the article that covers the OnDemand client version that you currently work with.

7.8 Globalizing applications by using ICU

If you want to enable your application to be multicultural support, consider using the ICU components. The ICU library is available for Java. The library is licensed under a nonrestrictive open source license that allows its use with both commercial software and other open source or free software.

Although portions of the Java framework in the packages java.text and java.util are already based on ICU code, ICU provides the following, among other, additional functionality:

- Code page conversions, which is the function ICU is used for in the Content Manager OnDemand infrastructure

  ICU conversion tables are based on the charset data that is provided by IBM. It can be considered one of the most complete collections of charset conversions available.

- String functions such as locale-based comparing and sorting

  The national conventions used by ICU are based on the Common Locale Data Repository (CLDR) project that is maintained by Unicode.

- Date and time formatting and calendar support based on the regional or locale settings

  This support includes translating month and day names into the selected language, choosing appropriate abbreviations and ordering fields correctly, and dealing with different calendars and time zones.

- Text analysis and regular expression support for Unicode data, as well as line wrapping and text boundary functions

You can download the ICU library and binaries for various platforms from the following Web page:

`http://www.icu-project.org`

For the Javadoc documentation for ICU4J, go to the following Web page:

`http://www.icu-project.org/apiref/icu4j`

8

Folder searching

In this chapter, we provide an overview of the techniques for performing Content Manager OnDemand (OnDemand) document searches by using OnDemand Web Enablement Kit (ODWEK) APIs. You can use the APIs in a variety of ways to search for stored documents. How the APIs are called depends on the nature of the desired search results.

This chapter covers the following topics:

- Criteria and SQL searches
- Sort fundamentals
- Search results
- Callbacks

8.1 Criteria and SQL searches

The ODWEK Java APIs provide two ways to search for documents. One way is called *criteria search*, which is also known as a *parametric search*. The other way is an SQL search.

8.1.1 Criteria search

A *criteria search* is a search that is performed on an OnDemand folder by setting search criteria field values and operators. The ODFolder object is searched by retrieving one or more ODCriteria objects (OnDemand folder fields) and setting the search values. A valid search operator must also be selected for each ODCriteria object. Each selected ODCriteria object has at least one search value. If the search operator is ODConstant.OPBetween (between) or ODConstant.OPNotBetween (not between), then the second search value must be set for the ODCriteria object.

Optimized search methods: The criteria search methods are *optimized* to provide the best possible performance. Use criteria search whenever possible.

In addition to the performance advantage, when performing a criteria search, the OnDemand application group database fields do not need to be queried. The criteria search is well structured and simple to use. All values in the criteria search are of type String. ODWEK converts the values to the proper format before it executes the search.

ODWEK uses dynamic statement caching for non-SQL based ODFolder.search API calls, which improves the performance of criteria search over the SQL search.

The SQL search requires advanced OnDemand and database knowledge to achieve correct results and good performance. The SQL search is also more complicated and requires use of the right data type for each application group field to be searched.

Follow these rules for preparing a criteria search:

1. Instantiate an ODFolder object with a valid OnDemand folder name.
2. Obtain a string array of ODCriteria names by using a call to the ODFolder.getCriteriaQueryOrder() method.

   This method returns the folder fields in the query order that the OnDemand administrator wants the search criteria to be entered and laid out.

3. For each ODCriteria String name that is returned:
   a. Instantiate the ODCriteria object by issuing the ODFolder.getCriteria(criteria_name) method.
   b. Obtain a list of valid search operators to display.
   c. Display or set the selected search operator to a valid search operator.

      `odCriteria.setOperator(oper);`
   d. Set one or two search values depending on the search operator.
   e. Perform the ODFolder object search.

Before performing a criteria search, consider the following points:

- Make sure the ODCriteria object is a folder field that can be queried as defined by the OnDemand administrator by calling the ODCriteria.isQueryable() method.
- Restrict the list of valid search operators to those as defined by the OnDemand administrator.

  Example 8-1 shows the code snippet for getting a list of valid operators for the ODCriteria object.

  Example 8-1 Code snippet to get a valid operator

```
int opers[] = odCriteria.getValidOperators();
String[] validOperatorNames = new String[opers.length];
for (int i = 0; i < opers.length; i++) {
   validOperatorNames[i] = ODUtils.operatorName(opers[i]);
}
...
public static String operatorName(int oper) {
   switch (oper) {
      case ODConstant.OPEqual:
         return ("Equal");
      case ODConstant.OPNotEqual:
         return ("NotEqual");
...
```

- For any Date field, make sure that the date pattern conforms to the default format for the folder field as defined by the OnDemand administrator.

  The default format for a Date field is `%m/%d/%y` or `mm/dd/yy`. Before the search page is written, the application must try to get the OnDemand folder date pattern, convert it to something more meaningful for users, for example, dd/mm/yyyy, and make a notation by the Date entry fields. Any date string that is received by the application must be converted to the pattern for each Date field that is specified for the folder before a search is performed.

Use the following code to obtain the default format:

```
String criteriaFormat = odCriteria.getDefaultFmt();
```

- Make sure that the search values are set for all required ODCriteria objects as defined by the OnDemand administrator for each folder field.

  Use the following code to check whether the ODCriteria object is required:

```
if (odCriteria.isRequired()) {
...
}
```

- The default operator setting for multiple ODCriteria search objects is AND. If multiple ODCriteria search objects are used, by default, they are ANDed together when the search is performed.

  The search can OR multiple ODCriteria search objects by calling and setting the ODFolder.setOrSearchCriteria() method to true. If an OR condition is required, the folder field chosen by the OnDemand administrator as the segment date is always ANDed with the rest of the collection of objects.

- Before presenting a search result list to a user, make sure that the display order of each ODCriteria name matches how the OnDemand administrator defined it. A value of zero (0) means that it is not eligible to display.

  Use the following code to check for the display order:

```
String[] displayOrderedFieldNames = odFolder.getDisplayOrder();
```

8.1.2 SQL search

You can write your application to use an SQL search. An SQL search involves setting the WHERE clause and invoking the ODFolder.search() method with the SQL string and additional parameters.

Always use the criteria search instead of an SQL search whenever possible. However, there might be times when a complex search is required, and an SQL search is the only way to provide correct search results.

Tip: As mentioned earlier, an SQL search requires *advanced* knowledge of OnDemand and the database to achieve correct results and good performance. Use the SQL search only when the criteria search is not sufficient for your search.

You can choose to search any field or predicate combination as long as you know the application group field names. A criteria search uses folder field names, and an SQL search uses application group field names.

Example 8-2 shows examples of WHERE clauses.

Example 8-2 WHERE clauses

```
WHERE balance BETWEEN 2000 AND 2020
WHERE name = 'IBM GIFT SHOP'
WHERE name = 'MARK''S SHOP'
WHERE name <> 'IBM GIFT SHOP'
WHERE crd_date > 10/03/95
WHERE name LIKE '%'
WHERE name > 'N' AND balance BETWEEN 2000 AND 2020
WHERE crd_date BETWEEN 01/01/2006 AND 12/31/2006
WHERE balance <= 100
WHERE name > 'N' AND (balance < 100 OR balance > 2020)
```

The application can dynamically query the OnDemand application group to obtain a list of field names as shown in Example 8-3.

Example 8-3 Application group query code snippet

```
List<String> listFields = new ArrayList<String>();
ODApplicationGroup applGroup =
odServer.getApplicationGroup(applGrpName);

for (Enumeration fields_enum = applGroup.getFields();
fields_enum.hasMoreElements();) {
   ODApplicationGroupField applField = (ODApplicationGroupField)
fields_enum.nextElement();
   listFields.add(applField.getName());
}
return listFields;
```

Each application group field can be displayed for the user to enter data values. After the matching values are received by the application, the query string can be created. The data type for each application group field must be known in order to create a correct SQL WHERE clause. The data type can be determined by calling the ODApplicationGroupField.getType() method.

An OnDemand folder can have one or more application groups mapped to it. There can be performance implications when a folder that is searched contains more than one application group. Setting the application group name can be accomplished by using the following API example:

```
odfolder.setApplGroupForSearchWithSQL("BaxterBayBank");
```

If the OnDemand folder has multiple application groups and this method is not called, then OnDemand performs a search across all application groups that are mapped to the folder.

When dealing with date values in the SQL search, the ODWEK Java APIs can handle either the default date format or a specified date format as shown in the following example:

```
String whereClause = "WHERE lname = 'DOE'";
Vector searchResults = folder.search(whereClause,"10/03/95",
"10/03/96","%m/%d/%y");
```

lname is the application group database field name. The two dates are the start and end of a date range for OnDemand to search using the date format that is specified as the last parameter in the method. An SQL search offers a more flexible search than a criteria search provides. When using a criteria search, you can query any of the folder fields that are available and specify the search concatenation between the selected fields to set as an OR condition or an AND condition. By using an SQL search, you can specifying the OR and AND conditions between each field or group of fields as desired.

Before performing an SQL search, keep in mind the following points:

- Use caution when doing an SQL search. Advanced knowledge of OnDemand and the database is required in most cases. Otherwise, you might not receive the best possible performance that the criteria search provides. Use a criteria search whenever possible.
- Identify the application group or groups that will be searched. Determine how many application groups are mapped to the Ondemand folder that you want to search by calling the ODFolder.getApplGroupNames() method. If there is more than one group, try to examine whether all of the application groups or just one group needs to be searched.
- Determine the application group field names that are needed to create the SQL query string. You can do this dynamically by using the APIs or a predefined method such as a properties or XML file.
- Determine the date range search, and set the date patterns according to the OnDemand date format that is specified by the OnDemand administrator.

- Before presenting a search results list to a user, make sure that the display order of each ODCriteria field matches the OnDemand administrator's folder field definitions. A value of zero (0) means that it is not eligible to display.

  Use the following code to obtain the display order:

  ```
  String[] displayOrderedFieldNames = odFolder.getDisplayOrder();
  ```

8.2 Sort fundamentals

When we speak about sorting, we address how the search result list is sorted and presented to the application. Sorting search results by the application is a seldom-used feature of the APIs and has inherent performance overhead for almost all of the API sorting techniques. Several API methods can be invoked to sort the result list in various ways. We discuss some of these techniques in this section.

The OnDemand administrator has the ability to define the default sort orders of the folder fields as they are displayed in the result list. This is not to be confused with the display order or query order sequences that are also defined for each folder field. We explain sorting by values within each folder field (or application group field).

In most cases, how a search result collection is sorted is closely tied to the number of maximum hits (maxhits) that can be returned for an OnDemand folder. The number of maxhits can be obtained by invoking the ODFolder.getMaxHits() method after a given ODFolder object is opened. This call returns either the maxhits value that is specified when the ODConfig object is instantiated or the value that is specified by the OnDemand administrator for the folder, depending on the lesser value. The number of maxhits can also be dynamically changed by invoking the ODFolder.setMaxHits() method.

For criteria searches, the search results are sorted as how the folder field sort order is specified, either by the API or by the OnDemand administrator. The application can explicitly set the sort order for each ODCriteria object by invoking the ODCriteria.setSortOrder() method. The value can be 0 to n, where 0 means that no sorting is specified. If all folder fields have their sort order settings equal to 0, then the search results are returned in the database order, which is based on random loads and optimizations.

No sorting: By default, *no* sorting is done in the ODWEK search routines, even if the OnDemand administrator has set a sort order on the folder fields.

The application can also specify where the sorting of the results occurs. The ODFolder.setSortLocation() method can specify the following sort order:

- ODConstant.OD_SORT_LOCATION_NONE

  The results (up to maxhits) are returned in no sorted database order.

- ODConstant.OD_SORT_LOCATION_MIDTIER

  The results (up to maxhits) are returned in no sorted database order and then are sorted by ODWEK native code.

- ODConstant.OD_SORT_LOCATION_SERVER

  All search results are sorted and then are returned on the OnDemand server, up to the maxhits in effect.

Let us assume that we specify the sort location as OD_SORT_LOCATION_MIDTIER and set maxhits to 200. When a search is performed and the actual number of results that met the criteria is 3000, the following actions occur:

1. The database query retrieves the first 200 rows that it finds that match the search criteria and then cancels the rest of the query. The results are optimized for database query speed and are returned in random order.
2. ODWEK (mid-tier) receives these 200 hits back and then sorts them.
3. A sorted list is presented, but you are not guaranteed to have the first 200 hits of the all possible hits that met the criteria.

If we apply the same assumptions as in the previous example, but change the sort location to OD_SORT_LOCATION_SERVER, then the following actions occur:

1. The database query is set to take the overhead to obtain all 3000 rows that match the search criteria,
2. The database sorts all 3000 rows.
3. The database returns only the first 200 rows from the sorted list to OnDemand.
4. ODWEK Java APIs pass the hitlist as is, with no additional sorting.

Depending on your application requirement, use the appropriate ODFolder.setSortLocation() method.

8.3 Search results

A search result list consists of the ODHit objects that represent an OnDemand document. The application uses the ODHit object to retrieve the documents as a byte array that can either be streamed directly to the requesting client or be written to a file on a shared file system.

8.3.1 Query and display order

Your application must check whether the OnDemand folder field can be queried, which can occur in several ways. After opening the ODFolder object:

- Call the ODFolder.getQueryOrder() method to get a string array of the folder field names in the order that they are defined by the OnDemand administrator for the folder. The string array has only the folder fields that have a non-zero query order value defined.
- After a list of ODCriteria objects is obtained from the selected ODFolder object, call the ODCriteria.getQueryOrder() method to get the value as defined by the OnDemand administrator. A value of zero means that the folder field is not eligible to be queried or searched.
- Call the ODCriteria.isQueryable() method. The method returns *true* if the folder field has a non-zero query value defined, or it returns *false* if the value is zero.

Your application must also check whether the OnDemand folder field can be displayed, which can be done in ways that are similar to checking whether the field can be queried. After opening the ODFolder object:

- Call the ODFolder.getDisplayOrder() method to get a string array of the folder field names in the order in which they are defined by the OnDemand administrator for the folder. The string array has only the folder fields that have a non-zero display order value defined.
- After a list of ODCriteria objects is obtained from the selected ODFolder object, call the ODCriteria.getDisplayOrder() method to obtain the value as defined by the OnDemand administrator. A value of zero means that the folder field is not eligible to be displayed in the results list.
- Call the ODCriteria.isDisplayable() method. The method returns *true* if the folder field has a non-zero display value defined, or it returns *false* if the value is zero.

8.3.2 Search result size

When your application allows users to perform their own searches, use care within the application to monitor the potential impact of the search. Users can inadvertently perform searches that create long running tasks on the OnDemand server.

The OnDemand administrator can set the maximum number of results that are returned during a folder search. Your application must honor this configuration setting or a lesser value when monitoring the impact of the search by users. For example, a call to the ODFolder.getMaxHits() method returns results based on either how the OnDemand administrator has the maximum hits value defined or how the ODConfig object property value is configured during the instantiation of the ODServer object, whichever is the lesser value.

After a client submits the search criteria to the application, a call to the ODFolder.searchCountHits() method can be made to determine the impact of the search. IBM DB2® has optimized APIs that are used to only get the count back for a specific query, but does not pass any data back. Therefore, overhead is avoided at both the OnDemand server and the mid-tier of setting up the hit structures. A decision in the application can be made based on the number of hits returned. For example, it can refuse the client search request or honor the request at another time of day.

If the maxhits size is not specifically set in the ODConfig object or the ODFolder object, or by the OnDemand administrator, ODWEK defaults the number of maxhits to 200.

8.3.3 Searching by date

The OnDemand administrator has the responsibility to define application groups for a collection of reports that will be loaded and indexed. One of the key database fields to be defined is a date field that represents either the actual date posted within the report content or the date that the report content is loaded into OnDemand. This field can also be designated as a segment field to OnDemand to improve overall performance when searching across the application group data tables.

Your application must add the date criteria as part of the client searches to eliminate potential performance issues on the OnDemand server. OnDemand converts all date formats into a common internal format when performing searches. Your application should check how a date field should be formatted when using it in a search. For example, the OnDemand administrator might define a date pattern for a folder date field as `%m-%d-%y`, which should be presented to the client as a search string pattern of mm-dd-yy. If no format is defined for the folder date field, it defaults to %m/%d/%y.

A call to the ODCriteria.getDefaultFmt() method returns the folder date pattern such as %m-%d-%y. Your application should ensure that the date search values follow the same pattern when performing a search.

8.3.4 Selecting a document from the search result list

After a folder search is performed, ODWEK returns the search result list as a vector of ODHit objects. Each object represents an OnDemand document. To retrieve a document from OnDemand by using the APIs, the ODHit object must be instantiated and then a call must be made to one of the ODHit.retrieve() methods to get the document returned as a byte array or file.

The document ID (docID) is a property within the ODHit object. It is used by ODWEK to retrieve the document content to the API. See the following example of a docID:

```
v7126-5011-5012-5014-BAA2-4FAAA-0-2652-0-1094-85-79-2-1-0-^A & T PIANO
C000000015213077277.860000
```

Different techniques are available to recreate the ODHit object that is used to retrieve the document. One technique is to persist the docID and recreate the ODHit object from the ODFolder object that has been persisted:

```
String docId = request.getParameter("docId");
ODFolder odFolder = (ODFolder) session.getAttribute("odFolder");
ODHit odHit = odFolder.recreateHit(docId);
```

This technique may not be desirable for applications that do not want to keep the docID persistent, perhaps in the browser search results HTML page.

Another technique is to obtain the ODHit object by its relative location within a collection of ODHit objects that has been persisted by the application or within a collection of ODHit objects that is recreated by calling the ODFolder.getHits() method. This method returns a list of ODHit objects from the last successful ODFolder search:

```
String hitPosition = request.getParameter("hitPosition");
ODFolder odFolder = (ODFolder) session.getAttribute("odFolder");
Vector odHits = odFolder.getHits();
ODHit odHit = (ODHit) odHits.get(Integer.parseInt(hitPosition));
```

8.4 Callbacks

With callbacks, you can handle search results on an item-by-item basis.

8.4.1 Why use callbacks

The methods that are described in the previous sections of this chapter use the default result handling behavior of the folder search API. The default behavior is simply to cache the results of the most recent successful search and make them available as a vector of ODHit objects after the search completes.

As of ODWEK version 8.4, the full query results are cached in the folder object, just as for a non-callback search. However, this caching behavior might change in future versions of the API to allow more memory-efficient result handling.

At present, searching with a callback provides the following benefits when you handle the result hits one by one:

- Provide users with a more dynamic experience by reporting and acting on partial results immediately.
- Cancel a running search.

8.4.2 Searching with callbacks

By using the ODWEK Java APIs, you can implement a custom result handler in the form of a *callback object*. You then call an ODFolder.search() method that takes your callback as a parameter. As the search executes, the API calls your custom handler for each hit as it is returned from the server.

Apart from invoking your custom callback logic, the behavior of callback searches match their non-callback counterparts. After you define a custom callback class, pass an instance of it to the ODFolder.search() method that matches your desired search type.

To implement a custom callback, extend the template class ODCallback, and override one of the following two methods that are provided for handling search results:

```
public boolean HitCallback (
   java.lang.String docId,
   char type,
   java.lang.String[] values)

public boolean HitCallback (
   java.lang.String docId,
   char type,
   int hit_location,
   java.lang.String[] values)
```

These two methods are the same except for the int hit_location parameter. The ODCallback class is not abstract. Therefore, you must only implement the method for the particular callback that you want to handle. All ODFolder search methods except the following one invoke the callback method without hit_location:

```
java.util.Vector search (
   ODCallback odCallback,
   boolean use_docloc_hit_callback)
```

Passing *true* for the boolean parameter causes the API to invoke the alternative HitCallback() method instead. The API passes a hit_location value that is one of the DocLocationXxx values that is defined in ODConstant. These values indicate the physical storage tier in which the referenced document resides. See the Javadoc documentation for ODConstant to view the possible values.

Despite the java.util.Vector return type of search() method signatures, callback searches return a vector of zero length, which the application code should ignore.

In addition to ODFolder's search() methods, there is an additional set of ODFolder methods named searchWithCallback(). Unlike the search() methods, searchWithCallback() calls are asynchronous. That is, they return immediately when called, and the query continues to execute on a different thread until completed or cancelled by the callback.

8.4.3 Callback search example

Example 8-4 shows a program that uses a custom callback to process search results. Each result hit is printed to the console as it is returned from the server, and the user is asked whether to continue searching for documents. The API passes result field values to the callback as a `string[]` of values in display order.

Example 8-4 Callback search example program

```
import com.ibm.edms.od.ODCallback;
import com.ibm.edms.od.ODConstant;
import com.ibm.edms.od.ODCriteria;
import com.ibm.edms.od.ODFolder;
import com.ibm.edms.od.ODServer;
```

```
/**
 * Demonstrates folder search using a custom callback object.
 */
public class CallbackSearchExample
{
   static ODServer odServer = null;

   public static void main (String[] args) throws Exception
   {
      connect (
         "CallbackSearchExample",
         "mydocserver", 1445,
         "myusername", "mypassword"
      );

      try
      {
         ODFolder folder = odServer.openFolder ("myfoldername");
         MyCallback callback = getCallback (folder);

         ODCriteria crit = folder.getCriteria ("FirstCriterion");
         crit.setOperand (ODConstant.OPEqual);
         crit.setSearchValue ("somesearchvalue");

         crit = folder.getCriteria ("SecondCriterion");
         crit.setOperand (ODConstant.OPLike);
         crit.setSearchValue ("%othersearchvalue%");

         folder.search (callback);
         folder.close ();
      }
      catch (Exception e) { e.printStackTrace (); }
      finally
      {
         disconnect ();
      }
   }

   static void connect
   (
      String applicationName,
      String server,
      int port,
      String usr,
      String pwd
```

```
) throws Exception
{
   odServer = new ODServer (new ODConfig ());
   odServer.initialize (applicationName);
   odServer.setPort (port);
   odServer.setServer (server);
   odServer.setUserId (usr);
   odServer.setPassword (pwd);
   odServer.setConnectType (ODConstant.CONNECT_TYPE_TCPIP);
   odServer.logon ();
}

static void disconnect () throws Exception
{
   if (odServer != null)
   {
      odServer.logoff ();
      odServer.terminate ();
      odServer = null;
   }
}

/**
 * Factory method
 *
 * @param folder An open ODFolder
 * @return A callback that prints displayable hit fields
 *         in correct display order for the given
 *         folder
 */
public static MyCallback getCallback (ODFolder folder)
{
   String[] displayOrder = folder.getDisplayOrder ();
   String[] queryOrder = folder.getQueryOrder ();

   int[] showIndex = new int[displayOrder.length];
   display: for (int i=0; i<displayOrder.length; i++)
   {
      ODCriteria criterion = folder.getCriteria (displayOrder[i]);
      showIndex[i] = -1; // -1 --> don't display

      if (criterion != null && criterion.isDisplayable ())
      {
         for (int j=0; j<queryOrder.length; j++)
            if (queryOrder[j].equals (displayOrder[i]))
         {
            showIndex[i] = j;
            continue display;
         }
      }
   }
   return new MyCallback (showIndex);
}

/**
 * Custom callback class that prints search results
 * to the console. Display values for hit fields
 * are printed in the order given to the constructor.
 */
public static class MyCallback extends ODCallback
{
   int[] displayOrder = null;

   public MyCallback (int[] showIndex)
   {
      displayOrder = new int[showIndex.length];
      System.arraycopy (
         showIndex, 0, displayOrder, 0, showIndex.length);
   }

   public boolean HitCallback (
      java.lang.String docId,
      char type,
      java.lang.String[] values)
   throws java.lang.Exception
   {
      // print index field values in display order
      for (int i=0; i<displayOrder.length; i++)
         if (displayOrder[i] != -1)
            System.out.print (values[displayOrder[i]] + " ");
      System.out.println ();

      System.out.print ("Continue searching (y/n)? ");
      String response = null;
      BufferedReader in =
         new BufferedReader (new InputStreamReader (System.in));

      try
      {
         response = in.readLine ();
      }
      catch (IOException e)
      {
         e.printStackTrace ();
         System.err.println (
            "Error reading user input; search cancelled.");
         return false;
      }

      if (response != null &&
         response.toUpperCase ().startsWith ("Y"))
      {
         return true; // true --> continue searching
      }
      else return false; // false --> cancel search
   }
}
}
```

9

Document retrieval

In this chapter, we discuss the way in which documents are retrieved by using OnDemand Web Enablement Kit (ODWEK) and how you use the plug-ins and transformations when retrieving documents. We pay special attention to Advanced Function Presentation (AFP) resources and the handling of large objects.

We cover the following topics in this chapter:

- The importance of a retrieval strategy
- Retrieval API overview
- AFP resource retrieval and custom caching
- Segmented retrieval and large object support
- Avoiding memory issues for large files
- Getting document type information
- Retrieving converted data

9.1 The importance of a retrieval strategy

Retrieving documents is a basic functionality of IBM Content Manager OnDemand (OnDemand) and is used frequently when users double-click hitlist items. The ODWEK Java APIs provide several methods for document retrieval. Most retrieval jobs can be done by calling the ODHit.retrieve() method, which retrieves the documents content and returns it as a byte array.

Retrieval can be complex when dealing with different data types and you have to present the data to users. For example, if you deal with line data, you do not want to show users a large retrieved spool as a big flow of Extended Binary Coded Decimal Interchange Code (EBCDIC) data in a browser. Instead, you want to use a line data viewer that is opened within the browser and let the viewer display the spool. Alternatively, you can use InfoPrint's AFP2WEB functionality to transform an AFP spool into a PDF document for viewing.

In addition to these conversions, special circumstances might require different handling and calling of different methods other than just a simple retrieve() method.

9.1.1 AFP documents

AFP documents do not consist of only single objects. They are a compilation of document data and resources such as fonts.

If you retrieve a complete AFP document, then the AFP resources are loaded into shared memory and are served from there for each request. If you deal with several different AFP documents or if you have large resources, this might not be optimal behavior. Likewise, if you retrieve multiple AFP documents to be processed in another application, you might not want the system to automatically load all the AFP resources. In these scenarios, you might want to deal with the AFP resources directly.

The OnDemand APIs provide methods to retrieve documents and rely on the resources being loaded into shared native memory. In addition, it is also possible to retrieve raw document streams and resource streams separately and have them written directly to file.

For more details about working with resources, see 9.3, "AFP resource retrieval and custom caching" on page 185.

9.1.2 Large objects versus small objects

You must treat large objects differently than small objects. Large documents must be differentiated in objects that have been loaded by using large object support, as explained in the following section, and large binary objects.

When retrieving documents that are large binary objects, for example, large PDF-files or large user-defined data types, it is important to consider memory usage. When you do a standard retrieval in a native format by calling the ODHit.retrieve() method, the system downloads the data from the OnDemand server and stores it in native memory to be delivered to the client. When working with large documents and many users, the system might run out of contiguous native heap. This issue can be addressed by bypassing memory and retrieving to file. See 9.5, “Avoiding memory issues for large files” on page 190, for more information.

9.1.3 Requiring only a small part of a large object

If you need only a small part of a large object, for example, a page from an large spool, you can use the ODWEK Java APIs to send a document in chunks.

Large spools that need to be transmitted in chunks have been archived by using large object support. OnDemand large object support is a feature that can be enabled to have large reports processed more efficiently by dividing it into segments. For example, a report that typically contains more than 1000 pages is divided into smaller chunks of 100 pages, for example. Each time a user views the document, only the requested chunk is retrieved and transferred. The API provides methods for the retrieval of specified segments.

For more details about dealing with segmented retrieval in general, see 9.4, “Segmented retrieval and large object support” on page 187. For details about how to enable the Line Data Applet in dealing with segmented objects, see 10.2, “ODWEK Java applets” on page 200.

9.1.4 Delivering documents

When designing an application, you can decide how a document shall be delivered by ODWEK, for example, as raw data stream, wrapped within an applet, converted to PDF, or viewed by using plug-in. The ODWEK Java APIs provide integrated data conversion for some types of documents. When retrieving a document, you can specify the format in which the data should be delivered. One way that is always possible is to have the document in its native raw format in which it is archived.

If you retrieve line data, for example, you can specify to use the included line data applet or have the spool converted to ASCII. When viewing AFP documents, you might want to have them directly converted into PDF by using the AFP2WEB Transform.

For more information about conversion and transformation, see 9.7, “Retrieving converted data” on page 193.

9.2 Retrieval API overview

The ODWEK Java APIs provide several methods for retrieving documents. Retrieval methods can be separated into two areas:

- Methods that retrieve documents
- Methods that retrieve resources (for AFP documents)

There are three ways to retrieve data by using the retrieval methods:

- Retrieval of the entire document
- Segmented retrieval of large-object documents in chunks
- Retrieval of the data to file instead of memory

All three ways are applicable for document retrieval. For resources, they are never segmented. Therefore, the segmented retrieval does not apply.

Most retrieval methods are placed in the ODHit class, which represents a document hit in a searched hitlist. Calling one of the retrieve methods within an ODHit object delivers the document that is represented by this ODHit instance.

In this section, we list and briefly explain all retrieval methods in the ODWEK Java APIs. For a detailed description and how to use them, see 9.3, “AFP resource retrieval and custom caching” on page 185, and 9.4, “Segmented retrieval and large object support” on page 187.

9.2.1 Retrieval APIs in the ODHit class

The retrieval methods that are in the ODHit class can be separated into three groups, which are methods that retrieve raw native document data, methods that support transformations (and applets), and methods that retrieve resources:

- Methods that retrieve documents in raw format:
  - `byte[] getDocument()`

    This method retrieves documents that are represented by the ODHit object in its native raw format as an array of bytes. For documents that have the large object flag enabled, only the first segment is returned.
  - `getDocument(java.lang.String filename, boolean allsegs)`

    This method retrieves the document in its native raw format and directly writes the data to a file. The `allsegs` parameter only applies to documents that have the large object flag enabled. For those documents, the parameter can be controlled if only the first segment or the entire document is retrieved.

  Tip: The getDocument methods always return the document data in native format. No conversion or transform is applied to the data, and no applet code can be returned. If you want to enable data conversions, use the retrieve method instead.

- Standard retrieval methods that support retrieval in different formats by using applets, transformations, or plug-ins:
  - `byte[] retrieve(java.lang.String viewer)`

    This method retrieves the complete document that is represented by the ODHit object by using the conversion format that is specified, for example APPLET, NATIVE, and PLUGIN. For documents that have the large object flag enabled, only the first segment is returned.

    Tip: You can specify an empty string as viewer parameter (retrieve(“”)). In this case, the viewer setting that is set in the ODConfig class during the initialization of the ODServer session is used. For more information about the ODConfig class initialization parameters, see “ODConfig parameters” on page 133.
  - `retrieve(java.lang.String viewer, java.lang.String filename)`

    This method works in the same way as the previous method, but writes the resulting data to a file instead of returning it as byte array. If large object support is enabled, only the first segment is written to file.
  - `byte[] retrieveSegment(int segment)`

    For documents that are retrieved in segments, this method provides access to the remaining segments after the first segment is retrieved by using the retrieve method.

  - `retrieveSegment(int segment, java.lang.String filename)`

    This method works similar to the previous method, except that it does not return the data as byte array. Instead, it writes the data to the file that is specified.

- Methods that deal with AFP resources:

  If an AFP document is retrieved by using the getDocument() methods, you must retrieve the resources separately, which can be done by using the following methods:

  - `byte[] getResources()`

    This method retrieves the AFP resource data for the document that is represented by this ODHit object and returns it as byte array.

  - `getResources(java.lang.String filename)`

    This method is similar to the previous method, except that it does not return the data as byte array. It writes the resource data directly to a file.

Example 9-1 shows how to use some of the retrieval methods that are provided by the ODHit class. For detailed information about how to use the retrieveSegment methods, see 9.4, “Segmented retrieval and large object support” on page 187.

Example 9-1 Retrieving documents

```
//Initialize using default configuration
ODConfig config = new ODConfig();
odServer = new ODServer(config);
odServer.initialize("/Applethandler");

/Connect to a server, open a folder and search
odServer.logon("9.156.238.77", "swelter", "qw1ert");
odFolder = odServer.openFolder("CustomerXX");
hits = odFolder.search();

if(hits.size() > 0) {
   //We are going to take the first hit
   odHit = (ODHit) hits.elementAt(0);

   //retrieve the document using default configured viewer:
   byte[] docdata = odHit.retrieve("");

   //retrive a LineDataApplet code for the document
   //Note that this will only for spools
   byte[] appletcode = odHit.retrieve(ODConstant.APPLET);
```

```
    //retrieve the native document and store it to a file:
    odHit.getDocument("C:\\file", true);
}
odFolder.close();
odServer.logoff();
odServer.terminate();
```

Example 9-1 does not include any exception handling or information about the initial configuration, which might lead to unforeseen results. See Chapter 6, "Connection pooling and connection handling" on page 111, for details about why exception handling is necessary and how to initialize and terminate connections.

9.3 AFP resource retrieval and custom caching

When working with AFP documents, a document does not consist of only a single object. Instead it is composed of resource data, such as fonts and overlays, and the actual data stream. When OnDemand archives AFP data, these resources are stored apart from the actual content. When retrieving documents that require specific resources for the first time, these resources are automatically loaded into shared native memory. When retrieving additional documents that use the same resources, the resources are directly accessed from shared memory.

Multiple resource groups: A single application can have multiple resource groups stored. Between loads to a single application, if the resources change in any way, for example they have different fonts or a new image, then a new resource group is loaded to that application.

Accessing the resources from shared memory enables a fast and efficient way to work with AFP documents, but it can have a significant effect on memory consumption. As mentioned in 6.5.3, "Allocation and release of resources and sessions" on page 136, shared resources are not released until all ODWEK client sessions (all ODServer objects including their dependent classes) are terminated. In environments in which you must deal with a large number of different reports, you might have a lot of different AFP resources that are loaded as shared memory in the machine that is running the Web application server. Because these resources can contain graphics, it is possible that the resource data for each report consist of several megabytes, which can lead to an increasing consumption of native memory over time.

Attention: You must ensure that your application correctly terminates all ODServer sessions and connections. For more information about how to correctly terminate an ODServer session and how ODWEK deals with shared resources, see Chapter 6, "Connection pooling and connection handling" on page 111.

If you work in such an environment, consider implementing a resource caching mechanism. By using the ODWEK Java APIs, you have the ability to retrieve document data and resource data separately. Table 9-1 lists the necessary API methods that are used to implement a custom caching mechanism.

Table 9-1 API functionality used in implementing custom resource handling

Functionality or action	API methods that are used
Search hitlist and select a document.	ODFolder.search(...)
Check if the document is an AFP document.	ODFolder.getDocType() returning ODConstant.FileTypeAFP
Get the internal ID of the resource data that is being used by this document.	ODHit.getResourceID()
Get the resource data that is necessary.	ODHit.getResources() or ODHit.getResources(String filename)
Get the data stream of the document.	ODHit.getDocument() or ODHit.getDocument(String filename, boolean allsegs)

A custom caching implementation can be implemented by storing the resources of all AFP documents that are requested by the user as files on the local file system of the Web server. If the resource of a requested document is already present as file, then it is served from disk. Otherwise the resource is retrieved from the server and stored on disk for subsequent requests.

The primary issue when implementing a custom resource caching system is to retrieve the resource data separately from the document data stream. You can retrieve the resource data by calling the getResources() method in the ODHit class. This method either returns a byte array that contains the resource data or writes the resources directly to a file.

If you consider implementing a caching-mechanism that uses files as storage for resources rather than memory or other storage, we recommend that you use the getResources(String filename) variation of the method. This variation bypasses memory and directly writes the raw data to the file system.

Having stored resource data is only the first part. You must also determine which resource you have in your cache, which you can do by using the getResourceID() method of the ODHit class. This method returns a string that uniquely identifies the resource that is being used by this document. The resource ID is a string that is constructed out of server name (IP), application ID, and AFP resource ID. Two ODHit objects that return the same resource ID use identical AFP resource data. Non-AFP documents or documents without any resources return *null* as a resource ID.

If you develop an application that deals more than just AFP documents, then check whether an ODHit really represents an AFP document, which you can do by using the ODHit.getDocType() method. This method returns a char. If the returned char equals the constant ODConstant.FileTypeAFP, then the document represented by this ODHit is an AFP document. You can continue to get the resourceID and check if you already stored resource data for this AFP. If you have stored the data, use the cached one. If you have not stored the data, retrieve the resources and store them.

9.4 Segmented retrieval and large object support

Large object support in Content Manager OnDemand provides enhanced usability and better retrieval performance for reports that contain large documents. For example, consider a report that contains statements that exceed 1000 pages. By using large object support, the statements can be divided into parts, for example, 100 pages.

When a user retrieves a statement, OnDemand retrieves and decompresses the first part of the statement. When the user moves from page to page of a statement, OnDemand automatically retrieves the part of the statement that contains the requested page as needed. Large object support can only be enabled with the AFP Conversion and Indexing Facility (ACIF), OS/390®, or OS/400® indexer.

When large object support is enabled, implementing Web applications with the ODWEK Java APIs implies the usage of segmented retrieval. If an object is archived with large object support enabled, the behavior of the ODHit.retrieve() and ODHit.getDocument() methods changes as outlined in Table 9-2 on page 188.

Table 9-2 Changes in API behavior when dealing with large objects

API function in ODHit class	Standard behavior (not a large object)	Behavior when dealing with large objects
byte[] getDocument()	Returns the entire document in native format.	Returns only the first segment of the document in native format.
getDocument(String filename, boolean allsegs)	Writes the entire document to file. The allsegs parameter is ignored.	Depending on allsegs, either the entire document or just the first segment is written to file.
byte[] retrieve(String viewer) for APPLET as viewer on line data	An applet HTML code is returned. The applet loads the data in one part.	An applet HTML code is returned. The applet loads the segments as they are needed.
byte[] retrieve(String viewer) for HTML or PLUGIN as viewer for AFP data	AFP2HTML applet code or plug-in raw data is returned for the entire document.	AFP2HTML applet is capable of dynamically loading segments. Plug-in handles segments on its own.
byte[] retrieve(String viewer) for other conversions handled by transforms (for example PDF)	The entire document data (converted) is returned.	ODWEK passes either a single document or a segment.

With the change in the behavior of the retrieval methods, additional methods are required to get the segments other than the first segment:

- `byte[] retrieveSegment(int segment)`

  This method retrieves a specified segment through its segment number.

- `retrieveSegment(int segment, String filename)`

  This method retrieves a specified segment (identified by segment number) directly to file, by bypassing native memory allocation.

- `getNumSegments()`

  This method returns the amount of segments for the document.

- `getNumPagesInSegment()` and `getNumPagesInLastSegment()`

  This method returns the number of pages of which each segment consists.

No getDocument method can return a specified segment. Only the `getDocument(String filename, boolean allsegs)` method allows the retrieval of a multi-segment document into a file by setting the allsegs parameter to *true*.

When dealing with large object spools, we are limited to the use of the retrieve(…) and retrieveSegment(…) methods. This use implies no real limitation as long as the retrieve methods require a viewer parameter to be specified.

9.4.1 Retrieving segmented documents

In 9.7, “Retrieving converted data” on page 193, we describe the different possibilities and conversions in detail. Regarding large objects and segments, the conversions change their behavior as shown in Table 9-2 on page 188.

Retrieving segmented documents in native format

If you implement an application in which you want to access different segments of a document in native raw format, you can use the retrieve() and retrieveSegment() methods and specify the ODConstant.NATIVE constant as a viewer parameter. To get the data segments, you make subsequent calls to the retrieveSegment() method to obtain other segments of the spool.

You must ensure that the retrieve(String viewer) method is always called first. For example, if you know that you need only the tenth segment, you must retrieve the first one by using retrieve(String viewer).Then you can retrieve the tenth segment by using the retrieveSegment() method.

Using the viewers on segmented documents

When viewing documents by using one of the OnDemand-provided browser viewers, such as the AFP browser plug-in, the AFP2HTML Java applet, or the line data Java applet, the viewers handle large objects by themselves. You do not have to do any retrieveSegment() calls in this case.

Like the OnDemand Windows client, the plug-in and the Java applets automatically retrieve only the pages that are viewed by the user if the document is a segmented large object. The viewer displays the first page, and then the user decides which segments the viewer must retrieve afterwards by scrolling to other pages.

Retrieving converted large objects

If you do not want to get the document in native format, but instead want to apply one of the available conversions to the document, the segmented document is handled by the converters and viewers.

For more details about the two applets and how to implement the callback class, see 10.2, “ODWEK Java applets” on page 200. For more information about dealing with the AFP2WEB Transform, see 10.3, “AFP2WEB Transform” on page 205.

9.4.2 Obtaining segment information

If you use the getNumSegments(), getNumPagesInSegment(), and getNumPagesInLastSegment() methods to query information about a large object document, make sure that you use them in the right order. If you call one of the methods before you retrieve the first segment of the document, they all return zero. You must make a call to the ODHit.retrieve() method first. Depending on the viewer settings, this method either retrieves the first segment or hands off the data to a transformation.

After you call the ODHit.retrieve() method, ODWEK internally has the data about the number of segments and pages that are available. Therefore, only calling the getNumSegments(), getNumPagesInSegment(), or getNumPagesInLastSegment() method after ODHit.retrieve returns the correct data.

Tip: To determine whether a document is a large object, you can use the ODHit.isLargeObject() method. If this method returns *true*, then calling the ODHit.retrieve() method returns either the first segment of data or lets the document handle it by a transform or conversion. After that, you can call the ODHit.getNumSegments() method, for example, to get more information about the segmentation.

9.5 Avoiding memory issues for large files

When you deal with large PDF documents or other large binary files, you cannot use the benefits of segmented retrieval, which is possible with large object support. Large object support is available only for documents that are loaded by using the ACIF or the OS/400 indexer. For more details, see 9.4, “Segmented retrieval and large object support” on page 187.

When you retrieve a document directly into a Java byte array by using either the ODHit.retrieve() or the ODHit.getDocument() methods, the documents are loaded into native memory of the server that is running ODWEK. Because each document requires a contiguous block of memory for being loaded, you can reach memory limits if too many documents or documents that are too large are retrieved.

The ODHit.retrieve() and ODHit.getDocument() methods also provide a way to write the document data to file instead of returning it as a byte array. In this case, the native part of the API can write the document data directly to a file. Therefore, less memory is allocated, and fragmentation problems can be prevented.

You can determine the size of a document before retrieving it by using the ODHitProperties object, which can be obtained by calling the ODHit.getProperties() method. The ODHitProperties object provides information such as application (group) name, load name, application group ID, and other internal information. It also has an ODHitProperties.getLength() method that returns the uncompressed length in bytes of the document.

You can use the length information to decide how to retrieve the document in your application. If it is a small document, you can save the overhead of writing it to file. For big documents, you must have the data written to file and then pass out the data by using the httpResponse stream without holding much data in memory.

Performance: When working with binary documents, you generally do not have to use the ODHit.retrieve() method. The advantage of the retrieve() method is the data conversions that are offered. See 9.7, “Retrieving converted data” on page 193 for information about the conversions.

Because the conversions are designed to handle line data and AFP documents, if you do not deal with these document types, use the ODHit.getDocument() method instead.

9.6 Getting document type information

The ODWEK Java APIs provide several methods for getting information about the document to be retrieved. For example, you can retrieve the application group name by using ODHit.getApplGrpName, retrieve the application name by using ODHit.getAppName, retrieve the location of the document by using ODHit.getDocLocation, and retrieve the internal DocID by using ODHit.getDocID.

As mentioned in 9.5, “Avoiding memory issues for large files” on page 190, you can obtain an ODHitProperties object through the ODHit.getProperties() method. The ODHitProperties class provides additional information such as application group ID, application ID, resource ID, load data, table name, and compressed and uncompressed document length.

If you want to take different retrieval actions, based on the type of a document, consider using one of the following document-type related methods that are provided in the ODHit class:

- getDocType()

  This method returns the type of the document in correspondence to what is configured as a document type in the application by using the OnDemand Administrator. The return can be any of the types AFP, BMP, GIF, JFIF, LINE, META, NONE, PCX, PDF, PNG, TIFF, USRDEF, SCS_EXT, DJDE, SCS (all represented by ODConstant.FileType* constants).

- getFileExt()

  This method returns a string that contains the extension that is set for a user-defined document type as stored on the OnDemand server. Alternatively, if the predefined document types are used, a corresponding extension is returned such as `afp` for AFP documents or `lin` for line data.

- getMimeType()

  This method returns the MIME type of the selected document. For example, for an AFP, it returns `application/afp`; for a TIFF image, it returns `image/tif`; and for a PDF, it returns `application/pdf`. This information is available when the ODHit object is created and does not require any further internal calls to the server.

- getViewExt()

  This method returns the extension of a document after conversion. If you retrieve an AFP document by using the AFP2PDF Transform, this method returns `pdf`. You can also call this method by giving a viewer parameter that corresponds to the viewer parameter that is used in the ODHit.retrieve() method to get the extension for different viewers and converters.

- getViewMimeType()

  This method returns the MIME type of the document after it is retrieved by using a conversion. For example, if you retrieve an AFP as a PDF by using an AFP2PDF Transform, this method returns `application/pdf`.

If you query data that is set to USERDEFINED as a document type in the application, the extension that is returned by the getFileExt() method is the configured one. The MIME type that is returned is `application/ondemand extension-field=`*`EXT`*, where *EXT* is replaced by the extension that you have configured. For example, for an application that is configured as USERDEFINED with the .xls extension, you see the results as shown in Example 9-2 on page 193.

Example 9-2 Document information about USERDEF documents

```
getDocType: U (which is ODConstant.FileTypeUSRDEF)
getFileExt: xls
getMimeType: application/ondemand extension-field=xls
getViewerType: 1 (which is ODConstant.VIEWER_BROWSER)
getViewExt: xls
getViewMimeType: application/unknown
```

Using the data type information

By using the information that is provided by the ODHit class about the data type of the document, you can implement type-sensitive document handling:

1. Check for the document type by using the getDocType() method.
2. Evaluate the returned constant. By doing so, you can run different actions for different document types as in the following examples:
   - For images, you can call a custom servlet handling the image data.
   - For line data and AFP documents, you can use the default conversion (ODConfig) or an explicit conversion.
3. Handle user-defined documents (ODConstand.FileTypeUSRDEF) in regard to the extension that is configured in the application.

   If you develop a Web application, you must send documents with the correct MIME type and a file name ending in the correct file extension that the browser can recognize. You can use the getFileExt() method to get the extension of a user-defined document type and construct a MIME type, such as `application/[file extension]`, which is recognized by most browsers.

Another way to handle different document types is to retrieve the document by using the preconfigured viewers by using the ODConfig configuration. Then decide on the MIME type of the converted format (getViewMimeType()) after retrieving the document. Alternatively, you can use the extension of the converted document (getViewExt()) to decide what to do with the document.

9.7 Retrieving converted data

As mentioned in 9.2, “Retrieval API overview” on page 182, the ODWEK Java APIs contain a set of methods to retrieve data and resources in their native raw format. They also contain another set of methods to retrieve data with the capability of applying conversions to the data while they are retrieved.

The following methods support data conversion:

- `byte[] retrieve(java.lang.String viewer)`
- `void retrieve(java.lang.String viewer, java.lang.String filename)`
- `byte[] retrieveSegment(int segment)`

  For large object spools only.
- `void retrieveSegment(int segment, java.lang.String filename)`

  For large object spools only

The retrieveSegment() methods: The retrieveSegment() methods can be called only after first calling one of the retrieve() methods. Because the retrieve() methods pass the viewer parameter, the retrieveSegment() methods do not need to specify the viewer information. The retrieveSegment() methods carry on the conversion that is specified in the first retrieve() method call.

For more information about large object support and segmented retrieval of spools, see 9.4, "Segmented retrieval and large object support" on page 187.

The two retrieve() methods require a string parameter called `viewer`. Depending on the value that you specify, the data is converted to different formats or displayed by using a special viewer. The effect on the data that you receive as a return value depends on the type of viewer conversion that you choose. The available conversions can be separated into three groups:

- Viewers

  Viewers do not transfer document data in any way. Instead they have information for calling a specific viewer that is transferred. OnDemand has three viewers delivered with ODWEK:

  – A line data Java applet to display line data
  – An AFP2HTML Java applet to display AFP documents in the Web browser
  – An AFP browser plug-in
- Transformations

  Data conversions are done on the documents. OnDemand supports the AFP2WEB technology and the Xenos transforms. By using AFP2WEB, AFP documents can be converted to other formats such as HTML or PDF. Xenos allows more configurable conversations from various formats including metacode, line data, and AFP to HTML, line data, XML, PDF, and so on. See 10.3, "AFP2WEB Transform" on page 205, and 10.4, "Xenos transforms" on page 211, for details.

- Native retrieval

  Retrieve data in the same format as it is stored. This viewer type is the only one that is supported by all data types.

9.7.1 Supported data conversions and viewers

All possible values for the viewer parameter are defined as constants in the ODConstant class. The following list provides a complete reference of all valid values that can be used as viewer or conversion:

- ODConstant.APPLET

  If you specify ODConstant.APPLET as viewer on line data documents, the returned data is HTML code that invokes the line data Java applet for line data spools. This viewer type cannot be used for any other data except line data.

- ODConstant.ASCII

  This value performs an ASCII conversion on the document to retrieve. It can be used for line data and AFP documents. The text that is returned is a representation of the original spool concerning text and layout, but it is converted and limited to an ASCII code page.

- ODConstant.HTML

  The ODConstant.HTML viewer conversion can only be used with AFP documents. The AFP document itself is converted to HTML by using the AFP2WEB Transform. Depending on the configuration of the AFP2WEB Transform, either the rendered HTML data is returned or the data is handled by using the AFP2HTML Java applet.

- ODConstant.NATIVE

  By specifying the ODConstant.NATIVE value, the document is returned in native raw format just as the getDocument() method does.

- ODConstant.PDF

  This value invokes an AFP2WEB transformation for an AFP document. In this case, the document is converted to a PDF document and returned.

- ODConstant.PLUGIN

  You can use this value when dealing with AFP documents only. It instructs the browser to use the AFP plug-in when displaying the AFP document.

- ODConstant.XENOS

  This value causes OnDemand to invoke the Xenos transformation.
- ODConstant.XML

  This value invokes an AFP2WEB transformation for an AFP document. In this case, the document is converted to an XML data document and returned.

For detailed information about how the applets, plug-ins, and transformations are used, see Chapter 10, “Applets, plug-ins, and transforms” on page 197.

10

Applets, plug-ins, and transforms

In this chapter, we describe the Java applet and browser plug-ins that are supplied with OnDemand Web Enablement Kit (ODWEK). In addition, we discuss transforms that can be integrated with ODWEK and how to configure ODWEK to use them.

We discuss the following topics in this chapter:

- ODWEK plug-ins
  - AFP plug-in
  - Image viewer plug-in
- ODWEK Java applets
  - Line data applet
  - AFP2HTML applet
- AFP2WEB Transform
- Xenos transforms

10.1 ODWEK plug-ins

ODWEK delivers several viewing applications to support Web applications that access IBM Content Manager OnDemand. Browser plug-ins are available for Microsoft Internet Explorer® and Netscape Navigator, as well as Java applets that can be used within any Java-capable browser and operating system. Because standard Microsoft Windows installations do not contain Advanced Function Presentation (AFP) viewing components and because you might want to provide enhanced usability for users when viewing line data, images, or AFP documents, consider using the provided applets or plug-ins.

The following list of plug-ins and applets is included in ODWEK:

- AFP plug-in

  This plug-in provides AFP document viewing capability in Web browsers.
- Image plug-in

  This plug-in provides enhanced support for image viewing in Web browsers. It includes support for PCX, BMP, and all other OnDemand image formats.
- Line data applet

  This applet provides line data spool display capabilities.
- AFP2HTML applet

  This applet provides AFP viewing capabilities by displaying AFP-HTML documents that are converted by AFP2WEB.

Compatibility: The plug-in viewers that are provided by IBM require Netscape Navigator 7.1 or later or Microsoft Internet Explorer 6.0 or later.

10.1.1 AFP plug-in

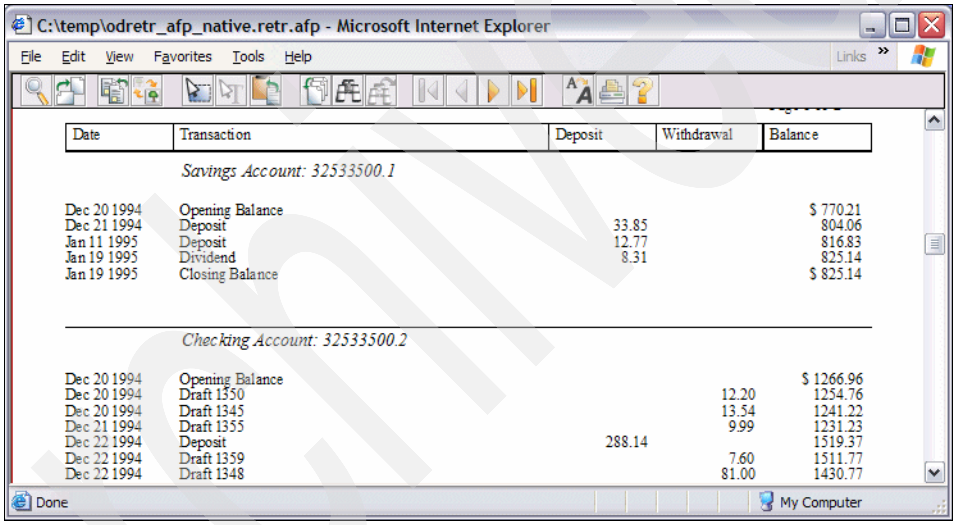
The AFP plug-in is a viewer for AFP documents that embeds into Web browsers. Like all other plug-ins, the AFP plug-in needs to be installed on all client computers that need to use it. The setup file is in the plug-ins subdirectory of your ODWEK installation. For more details about the directory structure and the functionality of the files that ship with ODWEK, see 1.5.3, “ODWEK Java API distribution files” on page 15.

If the AFP plug-in is installed on the client Web browser, it can be used to directly display all AFP documents in the browser.

If it is possible to distribute the plug-in to all client computers that require access to AFP documents, we recommend this option for display because it requires no conversion of data.

After users install the AFP plug-in, use the `ODHit.retrieve(ODConstant.PLUGIN)` method to retrieve the document, because the data stream is compressed and additional information for the plug-in viewer is passed. If you deal with large segmented AFP documents, data transfer is optimized because only necessary segments are retrieved by the plug-in. Also, all data that is transferred between OnDemand and the plug-in is compressed, which increases performance, especially on low bandwidth connections.

Figure 10-1 shows an AFP document as viewed in Microsoft Internet Explorer with the AFP plug-in.


C:\temp\odretr_afp_native.retr.afp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Links
Done
My Computer

Date	Transaction	Deposit	Withdrawal	Balance
	Savings Account: 32533500.1			
Dec 20 1994	Opening Balance			$ 770.21
Dec 21 1994	Deposit	33.85		804.06
Jan 11 1995	Deposit	12.77		816.83
Jan 19 1995	Dividend	8.31		825.14
Jan 19 1995	Closing Balance			$ 825.14
	Checking Account: 32533500.2			
Dec 20 1994	Opening Balance			$ 1266.96
Dec 20 1994	Draft 1350		12.20	1254.76
Dec 20 1994	Draft 1345		13.54	1241.22
Dec 21 1994	Draft 1355		9.99	1231.23
Dec 22 1994	Deposit	288.14		1519.37
Dec 22 1994	Draft 1359		7.60	1511.77
Dec 22 1994	Draft 1348		81.00	1430.77

Figure 10-1 Document view with the AFP plug-in

10.1.2 Image viewer plug-in

Like the AFP plug-in, the image viewer plug-in is installed on the user's Web browser on the client side. Its purpose is to enhance the image fidelity for users within the Web application. Most Web browsers are not capable of displaying TIFF, PCX, or BMP pictures, where the image viewer plug-in is capable of displaying all types of images that are supported by OnDemand (BMP, GIF, JFIF/JPG, PCX, and TIF). The image plug-in also provides additional functionality for viewing images such as zooming, rotating, adjusting the contrast or brightness, navigating through different pages, and scaling to gray to enhance text display.

Unlike the AFP viewer plug-in, the image viewer plug-in may not be necessary to deploy in certain environments. If you do not use image formats, such as PCX, and rely on JPG, GIF, or TIF, you might want to let the browser or the operating system handle the image files. All browsers are capable of displaying GIF and JPG images, and operating systems, such as Windows XP, are capable of viewing TIF or BMP images already.

10.2 ODWEK Java applets

The Java applets that ship with ODWEK provide significant help in the usability of Web applications. Where plug-ins must be deployed on client computers, applets are maintained on the Web server and are downloaded by the client Web browsers on demand.

The two applets that are available with ODWEK serve two document types:

- Line data applet

  The line data applet is developed to display line data. The applet itself provides no conversion. However, with the applet, you can enable Web users to view line data the same way as in the OnDemand Windows client. In addition to viewing a line data document, the applet also provides annotation functions. You can enable users to view and create annotations for the line data in a Web-based application.

- AFP2HTML applet

  The AFP2HTML applet works in a different way than the AFP plug-in. AFP documents are transformed to HTML by using the AFP2WEB Transform. After the transformation, the HTML content is displayed in the Web browser. If you want to provide additional features, such as segmented document retrieval, you can configure to display the converted AFP by using the AFP2HTML instead of letting the browser deal with the rendered AFP HTML.

Using the applets for viewing line data and AFP documents offers the following advantages:

- Users have nearly the same functionality with Web browsers as when they use standard OnDemand Windows clients.
- There is no need for any client installation. You do not need to install the applet anywhere, like you must do with the AFP viewer plug-in.
- The applets support segmented retrieval for large object documents. Only the segment that is currently being viewed by the user is retrieved from OnDemand. Other segments are retrieved whenever they are needed.

For more information about large object support and segmented retrieval, see 9.3, “AFP resource retrieval and custom caching” on page 185.

- The applets require Java version 1.4.1 on the client Web browser. Java is available for a variety of operating systems, including Windows, MacOS, and Linux.

In your Java application, provide a method that can be called by the applet. This method must pass the requests of the applets further on to the ODWEK Java APIs.

Figure 10-2 shows the link between the applet, your Web application, and ODWEK.

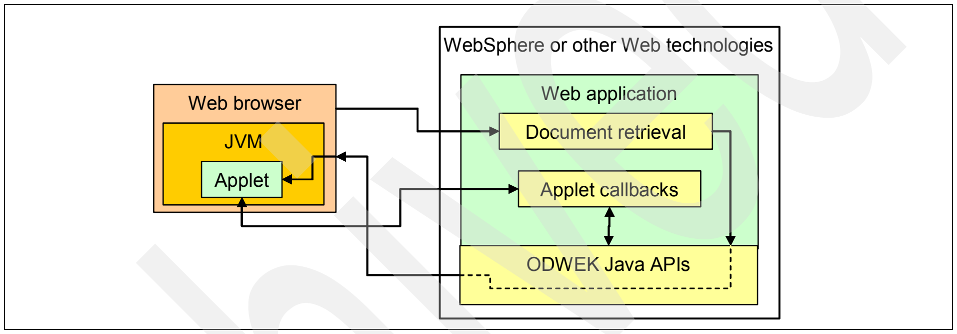


Figure 10-2 Communication scheme of the ODWEK Java applets

When a user requests a document from the Web application and the document retrieval code in the application decides to send the document as an applet, then ODWEK returns the HTML code, which embeds the applet invocation. With this HTML code, the browser can execute the applet in its Java virtual machine (JVM).

The data itself is requested by the applet through a callback methodology that must be implemented by the Web application. The callback method passes the requests to the ODWEK Java APIs. Within ODWEK, the applet requests are handled, and then the callback method delivers the results to the applet. Therefore, for the data stream, the callback method that your application implements is just a proxy that passes requests and data through.

Callbacks: Callbacks are not used for just applets. The AFP plug-in uses them as well for large object documents.

10.2.1 Configuring and using the ODWEK applets

As outlined in Figure 10-2, the ODWEK Java applets and the AFP plug-in require a callback methodology in your Web application if you want to use them. To implement this callback mechanism, you must provide a servlet that can be called by the applets and that passes the applets' queries to the OnDemand server.

The first step in implementing the applet callback servlet is to inform ODWEK about where to find the servlet. You do this in the initialization code of your application.

When calling the ODServer.initialize() method, specify a parameter called `applicationName`. This application name is the path to the applet callback servlet. For example, if you write an applet callback class called `ODViewerCallback` and make it available as `http://webserver/MyApp/ODViewerCallback`, then initialize the ODServer sessions by using `ODServer.initialize("/MyApp/ODViewerCallback")`.

The callback code of the ODViewerCallback servlet itself it straightforward. Pass the applet's request to the ODWEK Java APIs. The APIs provide the ODServer.viewerPassthru method, which is dedicated to this task. Therefore, the entire task that needs to be done is to get the request from the applet and invoke the viewerPassthru() method on your OnDemand session. Example 10-1 shows how to implement the applet callback code.

Example 10-1 Sample code to implement an applet callback

```
public void processRequest(HttpServletRequest request,
                              HttpServletResponse response) {
   try {
    ODServer odServer = null;
    byte[] results = null;
    HttpSession wwwsession = request.getSession(true);

    synchronized (wwwsession) {
         odServer = (ODServer) wwwsession.getAttribute("MyODSession");
         results = odServer.viewerPassthru(request.getQueryString());
    }
    OutputStream outputStream = response.getOutputStream();
    outputStream.write(results);
   } catch (Exception ex) {
      ...
   }
}
```

As you can see, the ODServer object is obtained from the session object, in which it is stored as MyODSession in the initialization and login code for this example. By placing this code in a servlet class and modifying how you get the ODServer object, you finish the implementation of the applet callback code. If you then use this servlet as the `applicationName` parameter in the ODServer.initialize() method, you meet all requirements for using the viewers.

Synchronization: Example 10-1 contains a *synchronized* code block. Synchronization is required when dealing with ODWEK sessions. See 6.4.2, “Synchronization” on page 129, for more information.

The servlet is automatically used for all viewers that call back to OnDemand.

The retrieve() method: When using the Java applets, you cannot use the retrieve() method with the file parameter. `ODHit.retrieve(ODConstant.APPLET, “c:\\file.ext”)` leads to an exception. You can only use the following retrieve() method, which directly returns the HTML as a byte array:

```
byte[] x = ODHit.retrieve(ODConstant.APPLET).
```

This applies to both line data and AFP2HTML applets.

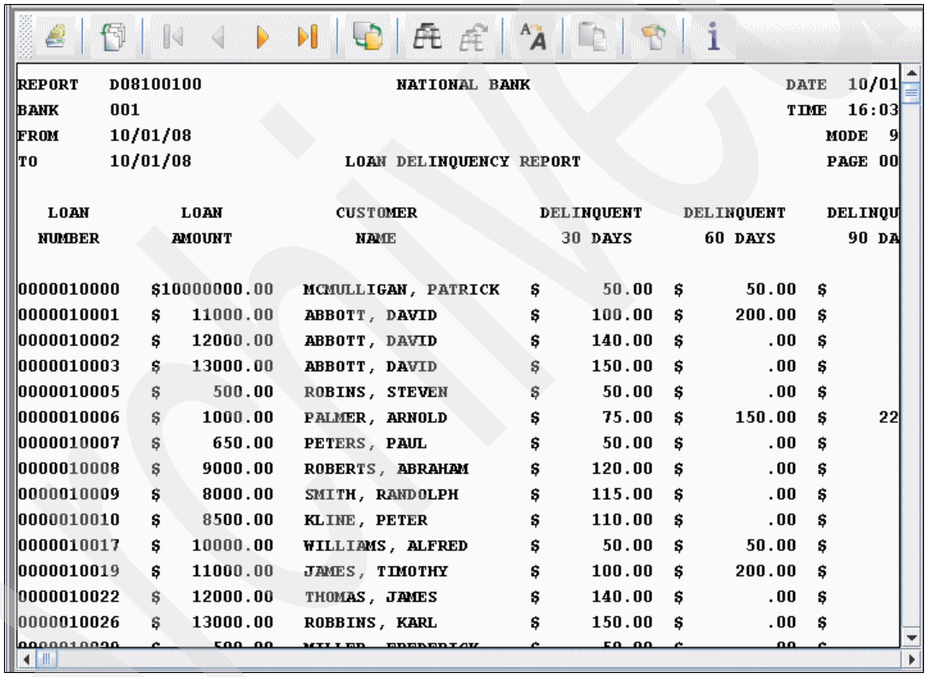
Compatibility: All applets that are supplied by ODWEK require a Java capable browser. All browsers that are currently available, for example Microsoft Internet Explorer, Netscape Navigator, and Mozilla Firefox, support Java. In addition, the applets require Java version 1.4.1 or higher.

10.2.2 Line data applet

When using the line data applet, you do not need to fulfill any special prerequisites except to have a callback servlet that passes the requests to OnDemand.

The applet, as shown in Figure 10-3, provides significant enhancements to the user in comparison to just passing the spools in the ASCII format (ODConstant.ASCII) to the browser:

- The user can view and add text annotations to the document.
- The applet provides a view on the spool that is identical to the view in the OnDemand Windows client. Document fidelity can be achieved in browser environments.
- The applet is capable of segmented retrieval, therefore, optimizing the document load for large objects.
- The applet provides some useful functionality, such as zooming, for better usage.



```
REPORT   D08100100                    NATIONAL BANK                         DATE  10/01
BANK     001                                                                TIME  16:03
FROM     10/01/08                                                               MODE  9
TO       10/01/08                LOAN DELINQUENCY REPORT                        PAGE 00

   LOAN          LOAN             CUSTOMER              DELINQUENT     DELINQUENT     DELINQU
  NUMBER        AMOUNT              NAME                 30 DAYS        60 DAYS        90 DA

0000010000  $10000000.00   MCMULLIGAN, PATRICK     $       50.00  $      50.00  $
0000010001  $    11000.00   ABBOTT, DAVID           $      100.00  $     200.00  $
0000010002  $    12000.00   ABBOTT, DAVID           $      140.00  $        .00  $
0000010003  $    13000.00   ABBOTT, DAVID           $      150.00  $        .00  $
0000010005  $      500.00   ROBINS, STEVEN          $       50.00  $        .00  $
0000010006  $     1000.00   PALMER, ARNOLD          $       75.00  $     150.00  $      22
0000010007  $      650.00   PETERS, PAUL            $       50.00  $        .00  $
0000010008  $     9000.00   ROBERTS, ABRAHAM        $      120.00  $        .00  $
0000010009  $     8000.00   SMITH, RANDOLPH         $      115.00  $        .00  $
0000010010  $     8500.00   KLINE, PETER            $      110.00  $        .00  $
0000010017  $    10000.00   WILLIAMS, ALFRED        $       50.00  $      50.00  $
0000010019  $    11000.00   JAMES, TIMOTHY          $      100.00  $     200.00  $
0000010022  $    12000.00   THOMAS, JAMES           $      140.00  $        .00  $
0000010026  $    13000.00   ROBBINS, KARL           $      150.00  $        .00  $
```

Figure 10-3 Line data viewer applet

10.2.3 AFP2HTML applet

The AFP2HTML applet is not just a viewing applet for AFP documents. It provides functionality to view a specially rendered HTML version of AFP documents. The job of rendering AFP documents to HTML documents is done by the AFP2WEB Transform. It works when AFP2WEB is correctly installed and configured. AFP2WEB is a special asset that can be integrated into ODWEK.

See 10.3, “AFP2WEB Transform” on page 205, for more information about AFP2WEB and how it can be configured.

After you have configured the AFP2WEB environment, to use the AFP2HTML applet, implement the callback servlet, if you have not already done this for the line data applet. Then retrieve the documents by using the ODHit.retrieve(ODConstant.HTML) method.

10.3 AFP2WEB Transform

The AFP2WEB technology is a transformation asset that can transform AFP data streams into HTML, PDF, and XML for Web browser viewing. ODWEK can use the AFP2WEB technology to transform AFP documents before they are sent to users. By transforming the documents, you can write Web applications that serve users who cannot use the AFP viewer or the AFP plug-in.

The AFP2HTML applet, which displays AFP documents by using a Java applet in the browser, also uses the AFP2WEB Transform to process information. Because the return of the AFP2WEB Transform is valid HTML, you can omit the AFP2HTML applet and directly pass on the HTML to the browser depending on how the transform is configured.

AFP2WEB licensing information: AFP2WEB is an asset of InfoPrint Solutions Company, formerly IBM Printing Systems Division. It is bundled for usage with Content Manager OnDemand. See your IBM representative for more information about the AFP2WEB Transform and further licensing details.

10.3.1 Configuring the AFP2WEB Transform

The binaries for using the AFP2WEB Transform to process AFP documents into PDF or HTML documents by using ODWEK are included in the ODWEK installation.

Note: The use of the AFP2WEB product requires additional entitlement.

Extracting the binaries

In your ODWEK installation directory, there are two subdirectories:

- afp2pdf contains the binaries for transforming AFP to PDF.
- afp2web contains the binaries for transforming AFP to HTML.

Both directories contain a compressed archive that must be extracted before the transforms can be used. You can extract the archives directly into their directories. You can also extract the binaries to any other place on the file system. Make sure that the directories are accessible by ODWEK. Also make sure that you extract the two transformations into two separate directories. They do not work when they are both integrated into one directory.

After you extract the AFP2WEB and AFP2PDF Transforms, they are ready to use. You can test the functionality by calling the executable by using an AFP document such as the one provided in the installation directories (insure.afp).

To test, enter either of the following statements on a command line, which results respectively in an insure.pdf and an insure.html file:

```
afp2pdf insure.afp
afp2web insure.afp
```

Configuration files

You must configure the transform that you want to use in ODWEK. You do this by using an INI file that is already present in the ODWEK directory. Open and modify the INI file that applies to the transformation that you want to use:

- afp2html.ini (AFP to HTML Transform uses afp2web binaries)
- afp2pdf.ini (AFP to PDF Transform uses afp2pdf binaries)
- afp2xml.ini (AFP to XML Transform)

Each INI file is built in the same way. For each application group and application pair in OnDemand, you can set different configuration values. In addition, a default section exists that applies to all applications that are not configured explicitly.

For each application, create an INI file section in the notation `[applicationgroup-application]`, where application group and application are separated by a hyphen (-). For application groups that contain multiple applications, you must have an explicit section for each application. Otherwise, the default configuration applies to a non-mentioned application.

To set the configuration values for the default configuration, you must have a `[default]` section. The configuration that you enter is used for all applications that are not explicitly configured. In some environments, it might be sufficient to have just the `[default]` section and nothing else, which causes all applications to use the same configuration.

Example 10-2 shows an afp2html.ini file. It is taken from the sample default configuration that ships with ODWEK.

Example 10-2 Sample afp2html.ini file

```
[CREDIT-CREDIT]
UseApplet=FALSE
ScaleFactor=1.0
CreateGIF=TRUE
ShadeFlag=FALSE
SuppressFonts=FALSE
FontMapFile=
ImageMapFile=c:\inetpub\scripts\ondemand\imagemap.cfg

[default]
UseApplet=TRUE
ScaleFactor=1.0
CreateGIF=TRUE
ShadeFlag=FALSE
SuppressFonts=FALSE
FontMapFile=
ImageMapFile=c:\inetpub\scripts\ondemand\imagemap.cfg
```

The sample file contains one explicit configuration for the application CREDIT of the application group CREDIT. For all other applications, the configuration values of the `[default]` section apply.

Configuring AFP2HTML and AFP2XML

The afp2html.ini and afp2xml.ini files both have the same configuration options, which can be set for each section. Table 10-1 outlines a short description of each option.

Table 10-1 AFP2HTML and AFP2XML configuration file settings

Configuration option	Description
ScaleFactor	Scales the output with the given scale factor. The default value is 1.0. The default size is derived from the Zoom setting on the Logical Views page in the OnDemand application.
CreateGIF	Indicates whether to create a GIF file for images.
UseApplet	Defines whether to use the ODWEK AFP2HTML Java applet to view rendered HTML pages or return the converted HTML output itself.

Configuration option	Description
AllObjects	Determines how ODWEK processes documents that are stored as large objects in OnDemand. The default value is zero (0), which means that ODWEK retrieves only the first segment of a document. If you specify one (1), ODWEK retrieves all of the segments and converts them before sending the document to the client.
ShadeFlag	Indicates whether to create shaded areas for all images.
SuppressFonts	Determines whether the AFP text strings are transformed. If you specify SuppressFonts=TRUE, any text that uses a font listed in the font map file is not transformed.
FontMapFile	Identifies the full path name of the font map file. The font map file contains a list of fonts that require special processing. See the AFP2WEB Transform documentation for details about the font map file.
ImageMapFile	Identifies the image mapping file. The image mapping file can be used to remove images from the output, improve the look of shaded images, and substitute existing images for images that are created by the AFP2WEB Transform. Mapping images that are common across your AFP documents, for example, a company logo, reduces the time that is required to transform documents.

UseApplet option: The UseApplet configuration setting controls whether ODWEK uses its AFP2HTML Java applet. When it is set to `TRUE`, the ODHit.retrieve() method returns an HTML code for executing the AFP2HTML applet when using the ODConstant.HTML viewer conversion. If you do not want to use the Java applet and instead want the actual HTML output that is generated by AFP2HTML, set the UseApplet option to `FALSE`.

For more detailed information about the configuration values that you can set in the INI files, see Appendixes F and G in the *IBM DB2 Content Manager OnDemand for Multiplatforms Ver 8.4: Web Enablement Kit Implementation Guide*, SC18-9231.

Configuring AFP2PDF

For the AFP2PDF transformation, only a subset of the AFP2WEB options is applicable. That is *ImageMapFile* and *AllObjects*, which are described in Table 10-1 on page 207.

In addition to these two common options, there is the *OptionsFile* option. The OptionsFile parameter identifies the full path name of the file that contains the transform options that are used by the AFP2PDF Transform. The transform options are used for AFP documents that require special processing. See the AFP2PDF Transform documentation for details about the transform options file.

Further reference and documentation

Further information and documentation is available that describes AFP2WEB in general, how to use the AFP2WEB command line executables and Java APIs, and how to integrate AFP2WEB into other applications. You can refer to the documentation and sample files in the java_api subdirectory of the AFP2HTML and AFP2PDF installation. Alternatively, you can refer to the product documentation, which you can obtain from InfoPrint Solutions Company.

10.3.2 Integrating the AFP2WEB Transform in ODWEK

After you have a valid and working installation and optionally customized the configuration INI files, you must integrate the AFP2WEB Transform into ODWEK. You do so by configuring ODWEK session and indicating the location of your installation and of your configuration file.

The configuration of the ODWEK session is done by using a set of properties that are set in the constructor of the ODConfig class. In addition to the standard parameters that are mandatory, you can create an instance of an ODConfig object by specifying an additional parameter of type *Properties*. In this Properties object, place the data that ODWEK must know about each transformation:

- Installation directory
- Path to the INI configuration file

Example 10-3 shows how to create an ODConfig object. You can then use this ODConfig object to initialize the ODServer object.

Example 10-3 ODConfig initialization for using the AFP2WEB Transform

```
Properties props = new Properties();

props.put(
   ODConfig.AFP2PDF_CONFIG_FILE, "c:\\opt\\afp2pdf\\afp2pdf.ini");
props.put(
   ODConfig.AFP2PDF_INSTALL_DIR, "c:\\opt\\afp2pdf");
props.put(
   ODConfig.AFP2HTML_CONFIG_FILE, "c:\\opt\\afp2web\\afp2html.ini");
props.put(
   ODConfig.AFP2HTML_INSTALL_DIR, "c:\\opt\\afp2web");
```

```
ODConfig odConfig = new ODConfig(
   ODConstant.PDF,
   ODConstant.APPLET,
   null,
   500,
   "c:\\temp",
   "ENU",
   "c:\\temp",
   "c:\\temp\\trace",
   4,
   props
);

ODServer serversession = new ODServer(odConfig);
```

For each transformation, a configuration constant is available for CONFIG_FILE and INSTALL_DIR, which both must be added to the Properties object. The resulting ODConfig object is used in the constructor of the ODServer class.

To use the AFP2WEB Transform, you can use the following viewer conversions as a parameter for the ODHit.retrieve() method. See 9.7, "Retrieving converted data" on page 193, for a more detailed description of how to use the viewer conversion constants.

- ODConstant.HTML converts the AFP to HTML.

> **AFP2HTML conversion**: When using the default configuration in the AFP2HTML configuration INI file, the ODHit.retrieve() method returns HTML code for executing the AFP2HTML applet when using the ODConstant.HTML viewer conversion. If you do not want to use the Java applet and instead want the actual HTML output that is generated by AFP2HTML, alter the afp2html.ini and change the UseApplet option to FALSE.

- ODConstant.PDF converts to PDF.
- ODConstant.XML converts to XML.

10.4 Xenos transforms

Xenos transforms can be used to apply transformations to AFP, line data, and Metacode documents when retrieving them from an OnDemand server. For example, you can use ODWEK to retrieve a Metacode document from the system, call the Xenos transform to convert the Metacode document, and send the converted output to the browser. The Xenos conversion is used with ODConstant.XENOS when calling the ODHit.retrieve() method.

When retrieving documents from the system by using ODWEK, you can use the Xenos transforms to perform the following tasks:

- Convert AFP documents to HTML, PDF, or XML files.
- Convert line data documents to AFP, HTML, PDF, or XML files.
- Convert Metacode documents to AFP, HTML, PDF, or XML files.

Important: Before you attempt to use the Xenos transforms on your system, you must obtain the transform programs, license, and documentation. See your IBM representative for more information. Also see your IBM representative for information about education and other types of help and support for installing and configuring the transform programs and processing input files with the transform programs.

10.4.1 Configuring ODWEK to use Xenos transforms

The Xenos transform is configured by using an INI file called *arsxenos.ini*. The file is placed in the ODWEK installation directory by default. You do not need to keep the default name or the location, because the API is instructed about the location of the INI file by the ODConfig object.

The arsxenos.ini file

The Xenos transforms can convert AFP documents into HTML, PDF, or XML output, and line data documents or Metacode documents into AFP, HTML, PDF, or XML output, that can be viewed from a Web browser. An administrator must specify the configuration options for the documents that Xenos transforms process.

The structure of the arsxenos.ini file is similar to the configuration files that are used for the AFP2WEB Transform. For each application and application group, a section must be created in the format `[applicationgroup-application]`. For all applications that do not have specific sections, the `[default]` section applies. Example 10-4 on page 212 shows a configuration for the CREDIT application group and the CSTATEMENTS application.

Example 10-4 The arsxenos.ini configuration section

```
[CREDIT-CSTATEMENTS]
ParmFile=/usr/lpp/ars/www/afp2pdf/sample.par
ScriptFile=/usr/lpp/ars/www/noindex.dms
LicenseFile=/usr/lpp/ars/www/dmlic.txt
OutputType=pdf
AllObjects=0
WarningLevel=4
```

The following options must be set:

- ParmFile

  This option specifies the full path to the file that contains the parameters that are used by Xenos to convert the documents.

- ScriptFile

  This option denotes the full path to the file that contains the script statements that are used by Xenos to create the output file.

- LicenseFile

  This option specifies the full path of a valid Xenos license file.

- OutputType

  This option specifies the output document type after a Xenos conversion. If the input document is AFP, you can set the output type to HTML, PDF, or XML. If the input document is line data or Metacode document, you can set this parameter to AFP, HTML, PDF, or XML.

- AllObjects

  This option determines how ODWEK processes documents that are stored as large objects in OnDemand. If you specify zero (`0`), then ODWEK retrieves only the first segment of a document. If you specify one (`1`), then ODWEK retrieves all of the segments and converts them before sending the document to the viewer.

  Delay for large object support: If you enable large object support for very large documents, users might experience a significant delay before they can view the document.

- WarningLevel

  This option determines how ODWEK handles the return codes from the Xenos transform. The Xenos transform sets a return code after each document is converted. Use this parameter to specify the maximum return code that ODWEK considers to be good and send the converted document to the viewer. For example, if you specify a value of four (4), then the return code that is set by the Xenos transform must be four or less. Otherwise, ODWEK does not send the converted document to the viewer. The default value is zero.

The main part of configuring Xenos is done in the ParamFile and ScriptFile by using conversion parameters and script statements. For information and samples about how these files should look like, see Appendix E, "Xenos transforms" in the *IBM DB2 Content Manager OnDemand for Multiplatforms Ver 8.4: Web Enablement Kit Implementation Guide*, SC18-9231, or the Xenos product documentation.

Configuring ODWEK for Xenos using ODConfig

Integrating a configured Xenos environment into an ODWEK-based Web application is much the same as integrating the AFP2WEB Transform. The difference is that your application must tell ODWEK where the arsxenos.ini configuration file is and the installation directory of Xenos.

The configuration of the ODWEK session is done by using a set of properties that are set in the constructor of the ODConfig class. In addition to the standard parameters, which are mandatory, you can create an instance of an ODConfig object by specifying an additional parameter of the Properties type. Example 10-5 shows how an ODConfig object be displayed. If you want to use the AFP2WEB Transform as well, you must include the AFP2PDF or AFP2HTML configuration values such as shown in Example 10-1 on page 207.

Example 10-5 ODConfig properties configuration for Xenos

```
Properties props = new Properties();

props.put(ODConfig.XENOS_CONFIG_FILE, "c:\\xenos\\arsxenos.ini");
props.put(ODConfig.XENOS_INSTALL_DIR, "c:\\xenos");

DConfig odConfig = new ODConfig(
   ODConstant.PDF,
   ODConstant.APPLET,
   null,
   500,
   "c:\\temp",
   "ENU",
   "c:\\temp",
   "c:\\temp\\trace",
   4,
   props
);

ODServer serversession = new ODServer(odConfig);
```

Further information

You can find information about how to integrate configure and use Xenos with OnDemand and ODWEK in Appendix E, “Xenos transforms” in the *IBM DB2 Content Manager OnDemand for Multiplatforms Ver 8.4: Web Enablement Kit Implementation Guide*, SC18-9231.

11

Document storing and updating

In this chapter, we deal with aspects of the OnDemand Web Enablement Kit (ODWEK) Java APIs that are not covered in the earlier chapters of this book.

We discuss the following topics in this chapter:

- Updating document indexes
- Storing documents
- Deleting documents

11.1 Updating document indexes

For a normal Web application that is delivered internally to users in a company or externally to users on the Internet, you do not want to have users update the index values of a document. However, there might be cases in which the update API must be used in an application.

11.1.1 Use cases for the update API

One example of using the update API might be for late indexing. For example, your application has to scan documents that come with barcodes. At the time of archiving the documents, you do not have additional information about them except the barcodes. In the evening or during an off shift, you can create a program that runs through the data in OnDemand and fill in all other values by using a database lookup on the barcodes.

Another example of using the update API might be for documents that are archived with references to SAP® business objects. Instead of writing a user exit program that is invoked during the load process, you can create a Java application that later queries additional data from SAP and updates the index values in OnDemand.

Depending on how you use OnDemand, you can also implement some custom applications by using the ODWEK Java APIs where you might want to change index values on a regular basis. For example, you might want to use a flag field to represent the current state of a document in a workflow.

11.1.2 Update methods in the ODWEK Java APIs

The ODWEK Java APIs provide the following two methods for updating database values for a document:

- `ODFolder.updateValuesForHits(Vector hits, Hashtable newValues)`

  Use this method from the ODFolder class if you want to update multiple documents at the same time and if you want to assign the same values to all of these documents.

- `ODHit.updateValuesForHit(Hashtable newValues)`

  Use this method from the ODHit class, if you want to work directly with the document that is represented by the ODHit instance. The database index values of this document are updated by using this method.

Both methods work by using a hash table, which contains field name and field value pairs. The field names that are used here are not the names of the database fields that you have set when creating the application group. Instead you must specify the field names as they are used in the folder through which this document is accessed. All fields and values should be specified to avoid errors.

Even when using the ODHit.updateValuesForHit() method, it internally passes the data to the ODFolder.updateValuesForHit() method. At this level, the fields of the application group are not known. Only the names that you gave the fields in the folder are known. Therefore, you must specify folder field names when you want to update database values.

The value to which you want to update a database field must be specified as a string. Despite the fact that a hash table can take all Java data types, storing a data type other than string in the hash table results in an exception. The strings are evaluated and converted automatically into the data type that is required by the application group. Similar techniques for converting strings are used in the generic indexer.

11.1.3 Hints and tips

Consider the hints in the following sections when writing applications that use the update functionality.

Date formats

When altering database values of the date format, you must specify the date in a standard U.S. format. That is, a date and time in the range '01/01/70' to '09/17/59' is required. Two-digit years less than 70 are interpreted as year 20*nn*.

Note: The date format, which is configured in the load information of the application (by using OnDemand administrator), is not evaluated here. You must use the U.S. date format.

Changing the DocID

ODWEK creates a unique DocID for each document. You can access it by using the ODHit.getDocId() method. The DocID can be used to internally reference the document or to directly retrieve the document without any new search required by using the ODFolder.retrieve() method.

The DocID is a constructed value that consists of server information, a load ID, resource data, and database field information. A sample DocID looks like the following example:

```
v7126-5018-5021-5030-EAA1-477FAA-0-0-0-26354-78-78-0-2-0-^   JPG   NEW
VALUE 1639883824   0.128.JPG 13882 13987
```

Each time you update database values by using the updateValuesForHit() methods, the second part of the DocID of the updated documents changes. If you use the DocID in any way in your application, make sure that you get the updated value using by the ODHit.getDocId() method after performing the database index update.

Use of the DocID: Because the DocID is constructed every time an ODHit is created (or the index field values of the hit are changed), do not use it as a persistent never changing value. Instead use it only as a short-term document reference.

Also, the DocID is an IBM internal data type, and it can change at anytime. Do not attempt to parse data from this structure. If data stored within the DocID is required by your application, see ODHitProperties.

Permissions

If you update the document database values, the user ID that you use must have the Document Update permission set in the application group.

Permission to update documents: Even if your user ID is a system administrator, you do not have permission to update documents. You must explicitly add this permission for each application group on the user ID or user group that you use in the application.

The permission is set by selecting the Update check box of the Document section on the Permissions tab of the application group properties in the Administrator.

You can check whether the user ID that you logged in with has the permissions to update database index values for a document by calling the ODHit.hasPermToUpdateDoc() method. This method returns a Boolean value that indicates whether the user can perform an updateValuesforHit() method.

11.2 Storing documents

OnDemand provides monitored directories and the ARSLOAD command to load data into the archive. Therefore, storing or archiving documents into an OnDemand system is usually a task that is not done by using the ODWEK Java APIs. However, in some use cases, you might need to directly store documents into Content Manager OnDemand by using the ODWEK Java APIs as an ad hoc method.

11.2.1 The storeDocument() method

The storeDocument() method provided by ODWEK to store documents is in the ODFolder class:

```
ODFolder.storeDocument(String path,
                       String appl_grp_name,
                       String appl_name,
                       String[] values)
```

Because the method is in the ODFolder class, you must first open a folder that maps the application group into which you want to store data. The parameters that are required by the storeDocument() method are self-explaining:

- `path` specifies the path to the file in the local file system on which the application is running.
- `appl_grp_name` specifies the application group name.
- `appl_name` specifies the application name.
- `values[]` specifies an array of strings that contain the database field values for the new document.

The array that contains the index values for the new document is a one-dimensional string array. The order in which the values are sorted in the array is relevant for the method. You must specify the values for the application group fields in the same order as they are displayed on the Field definition tab of the application group.

If you do not know which fields are needed in which order, you can evaluate the return of the ODFolder.getStoreDocFields() method. This method returns a two-dimensional array for the application or application group. The returning array contains either the application group field name (database field name) or the folder field name, in the same order in which you must create your array of new values for the storeDocument() method.

For example, consider an application group that has three fields: ARCHIVEDATE, CUSTNO, and RPTID. The fields are mapped to the three folder fields Archive date, Customer number, and Report ID. Calling the getStoreDocFields() method on this application group returns the array shown in Example 11-1.

Example 11-1 Sample output for getStoreDocFields

```
   Object[][] Fields = folder.getStoreDocFields("APPGRP", "APP");

Will result in:
   Fields[0][0] = ARCHIVEDATE
   Fields[0][1] = Archive date
   Fields[1][0] = CUSTNO
   Fields[1][1] = Customer number
   Fields[2][0] = RPTID
   Fields[2][1] = Report ID
```

When creating the array of the index fields for the new document, set the value for ARCHIVEDATE first, then for CUSTNO, and then for RPTID.

11.2.2 How the storeDocument() method works

The storeDocument() method works differently than the load mechanism in OnDemand. Calling a storeDocument() method stores the file into the specified application, but is not regarded as a load.

Load ID and system log

You do not receive a load ID for this process, and the system log contains a message that is different than what you normally get with a usual load. For each document that is stored by using the storeDocument() message, you see message #82, which is similar to the following example:

```
ApplGroup ObjStore: Name(PCFILES) Agid(5018) NodeName(QUSROND1) Nid(2)
Server(-LOCAL-) ObjName(551FAA) Time(3.425)
```

Load parameters and date format

The storeDocument() method does not use any indexer or load parameters. All default values, character removals, or date masks that you specify on the Load Information tab of the application group properties are ignored, because the storeDocument() method bypasses any load process.

If you must enter date values, do not specify them in the mask that you set on the Load Information tab. Instead provide the values in the form `%m/%d/%y` in the range from ‘01/01/70’ to ‘09/17/59’ where years less than 70 are interpreted as year 20*nn*. However, you can enter special characters such as `t` for the current date.

Note: The date format, which is configured in the load information of the application (by using OnDemand administrator), is not evaluated here. You must use the U.S. date format.

All other data is treated similar to the way in which the generic indexer converts strings into the corresponding database data type.

Expiration type

Because the storeDocument() method does not use a real load process, you cannot use this method on all application groups. For each application group, an expiration type is configured. The expiration type determines how the system expires and deletes data from the application group and can be of the following types:

- LOAD

  LOAD is the default expiration type. The system deletes an input file at a time from the application group. If the database organization is single load per database table, the system deletes a segment (table of index data and associated documents) at a time.

- STORAGE MANAGER

  The Object Access Method (OAM) or Virtual Storage Access Method (VSAM) storage manager deletes the data. The storage manager works with the ARSEXPIR program.

- SEGMENT

  The system deletes a segment (table) of data at a time from the application group.

- DOCUMENT

  The system deletes a document from the application group depending on the value of the Expire Data field.

Because no load process exists in storing a document, you cannot store any documents into an application group that has the expiration type set to LOAD. The application group must be set to either SEGMENT or DOCUMENT in order to enable the storeDocument() method. Calling the storeDocument() method with an expiration type of LOAD raises an ODException with the following message:

```
The server failed while storing a document.
```

11.3 Deleting documents

The ODWEK Java API that deletes documents is probably the least used API. Because OnDemand is designed as a long-term archiving system where deletion of documents is done through expiration times, deleting a document is not a function that is likely to be offered through an API or a custom application.

ODWEK contains the deleteDocs() method in the ODFolder class. This method requires a vector of ODHit objects. The purpose of the method is to remove the documents from the OnDemand database. The API is not designed to remove documents physically, which is a process that is often not possible, because data is aggregated into blocks and stored under control of WORM-media or an IBM Tivoli® Storage Manager system.

By calling the deleteDocs() method on documents, the database rows for these documents are deleted. The documents are no longer searchable, retrievable, or displayed on any hitlists. Also, because the exact position on the data blocks cannot be recovered, the document cannot be recovered in any way.

Permissions

By default, user IDs do not have permission to delete documents even if they are system administrators. You must manually set the permission in the application group for each user ID or user group. The permission is set by using the Update check box of the Document section on the Permissions tab of the application group properties.

You can check whether the user ID that you are logged in with has the permissions to update database index values for a document by calling the ODHit.hasPermToDeleteDoc() method. This method returns a Boolean value that indicates whether the user can perform an ODFolder.deleteDocs() call.

12

Memory and performance

Each OnDemand Web Enablement Kit (ODWEK)-based application is unique for tuning purposes because resource usage depends upon application design, the API features used, and the amount and pattern of API activity. In this chapter, we examine the performance tuning of applications that are written by using the ODWEK Java APIs with an emphasis on memory management.

We discuss the following topics in this chapter:

- Scope of performance tuning
- Memory
- Java heap
- The Java stack
- Garbage collection
- Startup parameters
- Other performance areas

12.1 Scope of performance tuning

The following subsystems most affect a given application's performance:

- CPU
- Memory (real and virtual)
- I/O (disk I/O, channels, and data paths)
- DASD (disk, optical, tape, other storage media including software interfaces)
- Network I/O (stacks and interfaces)

General tuning of these subsystems is beyond the scope of this book. However, it is important to plan enough capacity in each subsystem to support your application's anticipated usage. Also note that new Java releases frequently incorporate performance enhancements. Therefore, keeping your Java platform updated is an important element of performance management.

As Figure 12-1 shows, managing a multi-tiered environment involves managing these resources on all participating systems and network devices. Depending on the usage and application architecture, the ODWEK Java APIs run on any of the three tiers that are shown.

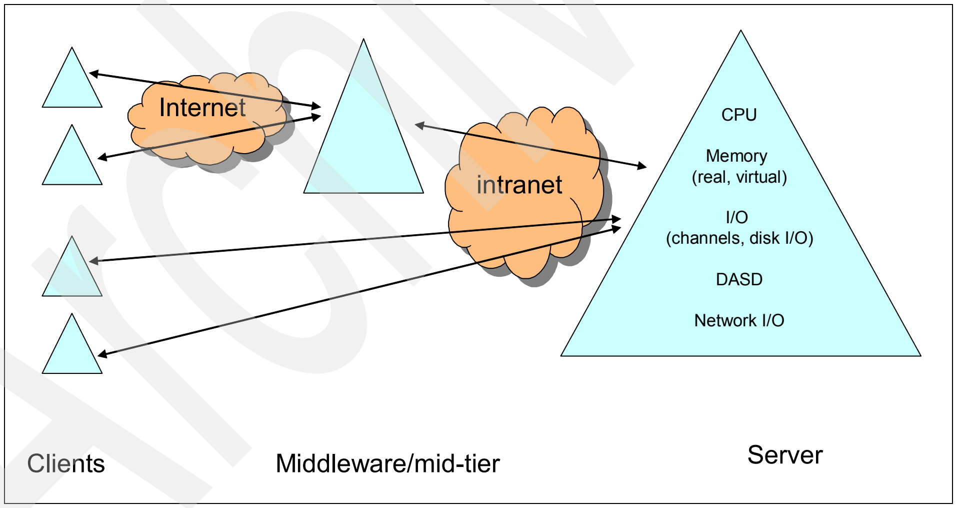


Figure 12-1 System performance tuning scope

12.2 Memory

The interface that is presented to applications by core Java libraries is a standard set by Sun™ Microsystems, but the internal workings of Java virtual machine (JVM) implementations differ by operating system, JVM release, and vendor. In the following sections, we avoid details or methods that are specific to particular platforms and Java implementations.

Note: JVMs are continually enhanced. See the documentation for your particular JVM for details concerning its internal operation.

Users of the IBM Java 5.0 implementation can consult the *Java Diagnostics Guide 5.0* at the following Web address for JVM profiling and troubleshooting information:

```
http://publib.boulder.ibm.com/infocenter/javasdk/v5r0/index.jsp?topi
c=/com.ibm.java.doc.diagnostics.50/diag/problem_determination/aix_me
mory.html
```

Virtual memory

The operating system maps portions of physical RAM and hard disk into a virtual address space, which applications see as one contiguous area of accessible memory. The physical location of program data changes as the operating system caches unneeded data to disk and loads currently needed data into random access memory (RAM). This is done in units called *pages*. This page-swapping system of *virtual memory* allows a group of running programs to consume more memory than there is physical real memory (RAM).

Figure 12-2 on page 227 illustrates the concept of virtual address space for a Java application. The size of the address space depends upon the hardware and operating system. The 32-bit systems can address a theoretical 4 GB of memory (2^{32} one-byte addresses). The 64-bit systems can address terabytes to exabytes of memory depending on the number of bytes per address.

The fraction of this address space that is available for use is often much less than the theoretical maximum. Recent 32-bit Windows versions, for example, allow programs to allocate up to 2 GB of memory, reserving 2 GB for the operating system. Overhead and allocation inefficiencies reduce the usable amount to less than 2 GB. Some 32-bit UNIX types allow programs to allocate roughly 3.75 GB.

Memory allocation

The JVM allocates native resources on behalf of running Java applications. Directly or indirectly, all Java programs consume four categories of allocated memory: native heap, native stack, Java heap, and Java stack.

The *native heap* comprises memory that the operating system allocates to a process. The amount of memory available corresponds to the usable portion of the virtual address space. The Java heap is allocated from this space.

A *native stack* is allocated by the operating system for each running thread. Each Java thread has a corresponding native thread and resources that are requested from the operating system by the JVM. Native libraries that are invoked through the Java Native Interface (JNI™) can create their own threads independently of Java.

The JVM requests *Java heap* memory from the operating system. The JVM makes this space available for the storage of data that pertains to Java class and array instances. The Java heap consumes a fraction of the total memory that is available to the running JVM process or rather a fraction of the native heap.

A *Java stack* is maintained for each of a Java program's executing threads. It stores thread-specific and method-local variables. A frame is allocated and pushed onto this stack when the executing code enters a Java method and is removed from the stack when a method returns.

Although it is common to speak of the Java and native heaps as though they were separate entities, the Java heap is simply that portion of the available native heap space that has been reserved by the JVM for this purpose. Most JVMs require that the entire Java heap be allocated as one contiguous (that is, unbroken) range of native memory addresses. Operating systems load native libraries and other data into various portions of the process' address space, which fragments the address space. When this happens, the maximum Java heap size is reduced to the largest remaining contiguous block.

Figure 12-2 shows the memory usage by Java applications.

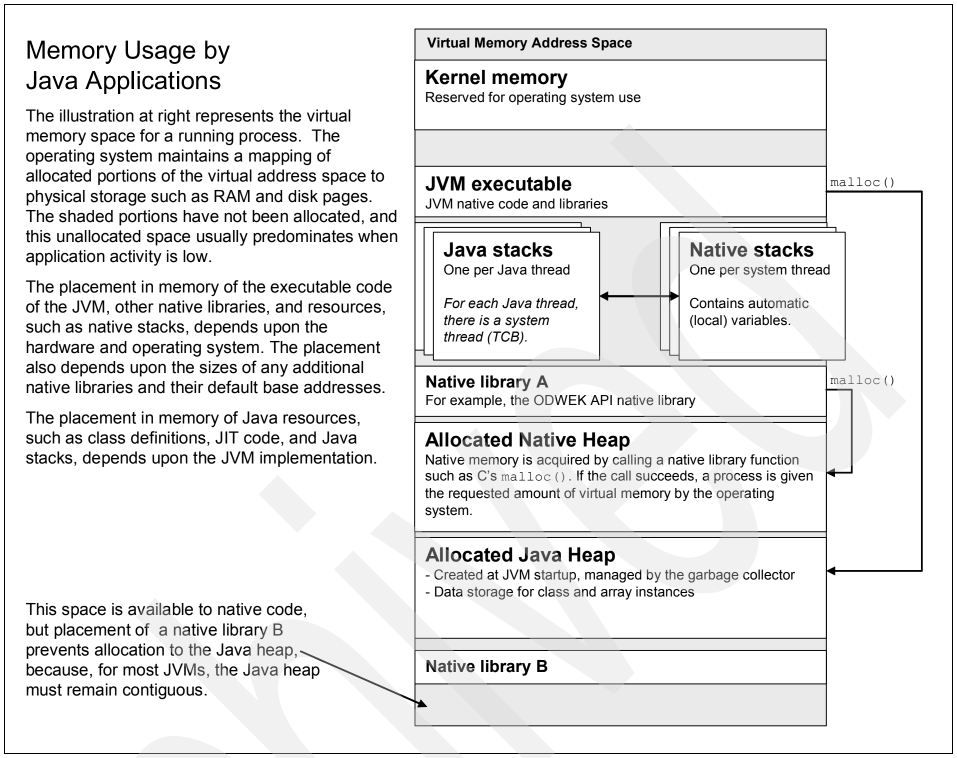

Memory Usage by Java Applications
The illustration at right represents the virtual memory space for a running process. The operating system maintains a mapping of allocated portions of the virtual address space to physical storage such as RAM and disk pages. The shaded portions have not been allocated, and this unallocated space usually predominates when application activity is low.
The placement in memory of the executable code of the JVM, other native libraries, and resources, such as native stacks, depends upon the hardware and operating system. The placement also depends upon the sizes of any additional native libraries and their default base addresses.
The placement in memory of Java resources, such as class definitions, JIT code, and Java stacks, depends upon the JVM implementation.
This space is available to native code, but placement of a native library B prevents allocation to the Java heap, because, for most JVMs, the Java heap must remain contiguous.
Virtual Memory Address Space
Kernel memory
Reserved for operating system use
JVM executable
JVM native code and libraries
Java stacks
One per Java thread
For each Java thread, there is a system thread (TCB).
Native stacks
One per system thread
Contains automatic (local) variables.
Native library A
For example, the ODWEK API native library
Allocated Native Heap
Native memory is acquired by calling a native library function such as C's malloc(). If the call succeeds, a process is given the requested amount of virtual memory by the operating system.
Allocated Java Heap
- Created at JVM startup, managed by the garbage collector
- Data storage for class and array instances
Native library B
malloc()
malloc()

Figure 12-2 Memory usage by Java program

The native API library in ODWEK, which is written in C, provides standardized access to the Content Manager OnDemand (OnDemand) server over TCP/IP. This mode of access is common to all Content Manager OnDemand clients.

ODWEK-based applications are Java programs that access native code by using the JNI. As such, depending on the implementation, they might place additional demands on the native heap separately from the JVM and Java heap. Optimizing the memory usage of an ODWEK-based application involves maximizing the amount of physical real memory (RAM) that is available to the running process. It also involves balancing the fraction of this memory given to the Java heap against the fraction that is remaining for use by the ODWEK native code. Reducing the Java heap size frees memory for use by native components.

12.2.1 Optimizing native memory

Strategies for optimizing native memory vary by hardware platform and operating system. In the following section, we discuss a few areas to consider.

Virtual memory

Virtual memory's swapping of pages to disk degrades performance dramatically. If possible, do not allocate more memory than there is physical real memory (RAM) in the system. In addition, avoid running other memory-intensive applications on middle-tier machines that run ODWEK applications. Set equal sizes for the operating system's minimum and maximum page file sizes to minimize page-file management.

Native libraries

The more native libraries that are loaded by a JVM process, the more likely it is that the process' native memory is fragmented. When possible, run ODWEK-enabled applications in their own JVM process. Also, avoid loading other native libraries into your ODWEK applications JVM process.

On Windows systems, if other shared libraries (dynamic link library (DLLs)) must share a JVM process with ODWEK, sometimes it is possible to "rebase" those other DLLs (not the ODWEK DLLs). By doing this, they occupy "out-of-the-way" addresses in a virtual address space of a process. The object of rebasing DLLs is to force them to occupy either very high or very low addresses, thus preserving the maximum contiguous block in the middle.

Allocating memory: When allocating memory, make sure that enough native memory remains *unallocated* to the Java heap to accommodate the volume of data that flows through the native layer of the APIs.

Working with large amounts of native memory means working with a smaller Java heap. In the following sections, we take a closer look at the Java heap allocation and garbage collection and how to tune both.

12.3 Java heap

The Java heap, which is illustrated in Figure 12-3 on page 229, is allocated as a contiguous block of system memory when the JVM starts. The Java heap is used mainly to store instances of Java arrays and classes. Java programs can create new objects as long as there is enough free memory for them in the Java heap or the heap can be expanded enough to accommodate them.

There is a single Java heap per JVM, and a single JVM is instantiated per process. Thus, the Java heap is shared by all threads that are running within a process.

Note: The developer must ensure that access to object instances by multiple threads does not result in unexpected behavior. See 6.4, "Thread safety" on page 128.

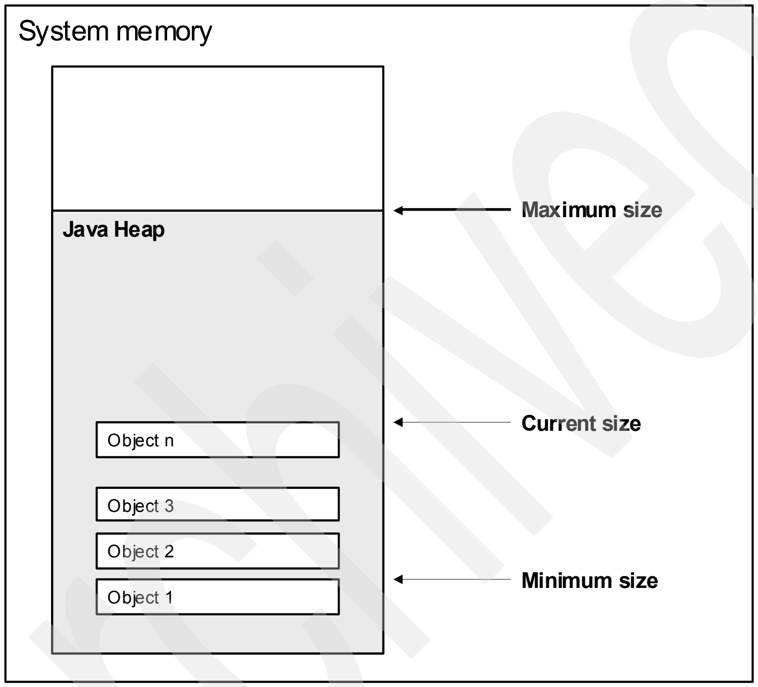


Figure 12-3 Java heap allocation

The default size of the Java heap differs by operating system. Table 12-1 gives the default values for some popular operating systems. These default sizes are subject to change. Check the documentation for your environment and Java version. These sizes and other preferences are adjusted by startup (command line) parameters, which are discussed in 12.6, “Startup parameters” on page 236.

Table 12-1 Default heap sizes

Operating system	Initial heap size	Maximum heap size
Windows	4 MB	Half the real storage with a minimum of 16 MB and a maximum of 2 GB-1 byte
Linux	4 MB	Half the real storage with a minimum of 16 MB and a maximum of 512 MB-1 byte
Solaris (32 bit)	3670 KB	64 MB
AIX	4 MB	64 MB
zOS	1 MB	64 MB
IBM i (formerly IBM i5/OS) Classic Java	16 MB	240 GB
IBM i (formerly i5/OS) Java J9	9 MB	2 GB

12.4 The Java stack

Each thread that is created has its own private Java stack. This stack includes all thread-specific variables, local variables, intermediate results, method invocation, and return data.

Stack memory is allocated in *frames*, each of which represents one method call. Nested method calls have the effect of pushing multiple frames onto the stack, which is why a stack trace partially exposes a program’s flow of execution. Some JVM implementations support parameters that control initial stack size.

Memory allocation failures that involve a thread’s stack throw the following types of errors:

- If a thread requires a larger Java stack than the maximum stack size, then the JVM throws a StackOverflowError.
- If the JVM has insufficient memory to create or expand the Java stack, then the JVM throws an OutOfMemoryError.

12.5 Garbage collection

Applications continually create and release object references. At some point, attempts to allocate new object instances begin to fail. Or a usage threshold is reached, and the JVM attempts to deallocate unused objects and return memory to the Java heap. The *Garbage Collector* is the agent that is responsible for deallocating unused memory and compacting the used memory in the Java heap. This process increases the contiguous free memory in the Java heap.

When a given object can no longer be reached through references that are held by active object instances, the JVM marks the unreferenced object as unneeded, that is, as garbage. The garbage collection process removes garbage objects that are marked as garbage from the heap and compacts the memory that is used by the remaining objects.

12.5.1 Garbage collection phases

There are several implementations of garbage collection, which vary in sophistication. You can use different internal architectures with the objective of optimizing performance for different types of application workload, memory configuration, and number of processors.

A single-threaded (serial) Garbage Collector is the most widely used architecture and can satisfy the memory management needs of most applications. We discuss the garbage collection method in this chapter. Popular tuning parameters are explored in 12.6, “Startup parameters” on page 236.

It is rare that significant garbage collection delays occur after tuning the serial Garbage Collector. However if garbage collection performance problems persist, consider using a more sophisticated garbage collection implementation. Large, multi-threaded applications that run on multiple processors and use large amounts of memory can benefit. See your Java vendor’s documentation for other garbage collection options.

As illustrated in Figure 12-4 on page 232, garbage collection is a three-step process:

- In the *mark phase*, all referenced objects are identified, and each object is scanned for references to other objects. The result is a vector that contains all reachable objects.
- An allocated-object vector is maintained by the JVM at object creation time. In the *sweep phase*, the reachable-object vector is compared with the allocated-object vector, and unreachable objects are deallocated.

- The *compaction phase* removes unused memory segments from between allocated blocks, resulting in a larger amount of contiguous available memory.

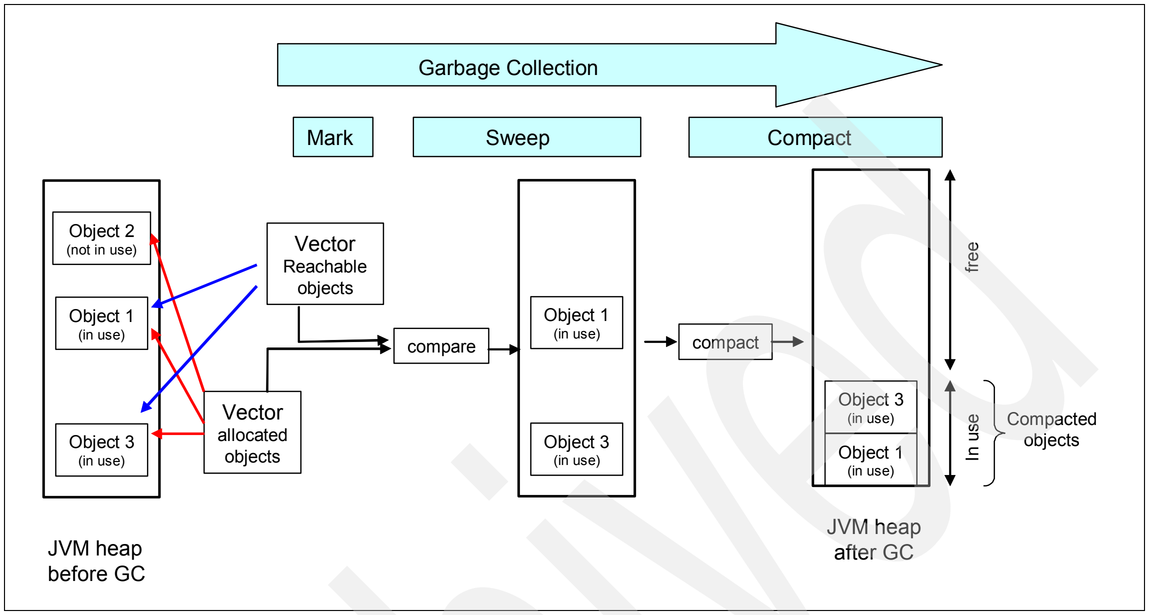


Figure 12-4 Garbage collection process: Mark, sweep, and compact

12.5.2 Garbage collection performance

No command explicitly deallocates memory for a Java object instance. Java code can request garbage collection by calling the System.gc() method. Depending upon the garbage collection implementation, System.gc() causes Garbage Collector to run immediately or at some point in the near future. When Garbage Collector runs, it takes a certain minimum amount of time. Unnecessary System.gc() calls can degrade performance. The precise effect depends upon the garbage collection implementation and your application's object creation and release behavior.

While the Garbage Collector is running, the JVM appears to be frozen. This is because the JVM must lock all objects in memory, preventing access to them while Garbage Collector runs. The duration of the garbage collection process is a function of the number of objects in the heap, especially the number of live objects. The larger the heap is, the more likely more objects will exist, and therefore, the longer the garbage collection process will run. The number of processors also affects garbage collection time. That is more processors require a longer collection time.

Tuning the garbage collection process: Tuning the garbage collection process is a priority for applications that maintain large heaps or run on multiple processors.

As illustrated in Figure 12-5, garbage collection (GC) for a large heap runs longer but less frequently. Garbage collection on a small heap runs faster but more often. While large and small are relative terms, the ideal size depends on your use of the ODWEK Java APIs.

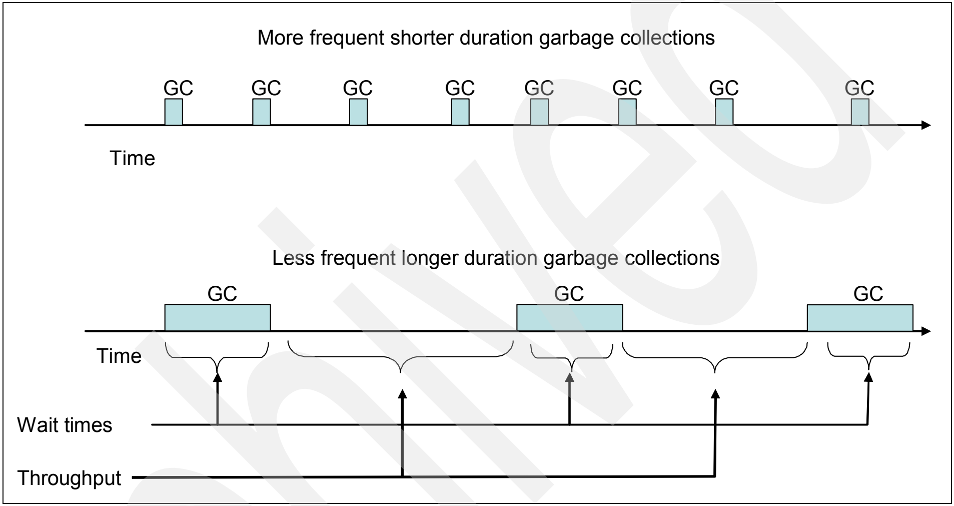

More frequent shorter duration garbage collections
GC
GC
GC
GC
GC
GC
GC
GC
Time
Less frequent longer duration garbage collections
GC
GC
GC
Time
Wait times
Throughput

Figure 12-5 Garbage collection frequency and its impact on wait time and throughput

Garbage collection performance is quantified by throughput, wait time, memory usage, and memory availability.

Throughput

Throughput is the percentage of time spent on activities other than collecting garbage over a long time period. Throughput includes time spent in allocation of memory and other memory management activities.

Wait time

Wait time is the percentage of time that an application appears unresponsive because garbage collection is running.

Memory usage

Memory usage counts the memory that is allocated while a process is running. Limited physical memory negatively affects system scalability.

Memory availability

Memory availability refers to the difference between the time an object becomes unreferenced and the time the memory becomes available for reuse. The memory that is used by unreferenced objects is released only after a garbage collection cycle has completed. The longer the period is between garbage collection cycles and the larger the heap is, the longer memory is consumed by unreferenced objects.

When the heap is exhausted, Java throws an OutOfMemoryError:

```
java.lang.OutOfMemoryError <<no stack trace available>>
```

A practical heap allocation strategy is to set the heap size large enough that you do not run out of memory. However, do not set the heap size so large that there is room to store too many marked-as-garbage objects before a garbage collection occurs.

Generational garbage collection

Simple garbage collection examines every live object in the heap. *Generational garbage collection* uses the observation that some objects have short lifetimes, other objects have long lifetimes, and still others survive for the lifetime of the JVM. To take advantage of this, the heap is divided into three separate areas as illustrated in Figure 12-6. These areas are called the *young*, *tenured* and *permanent generation* spaces.

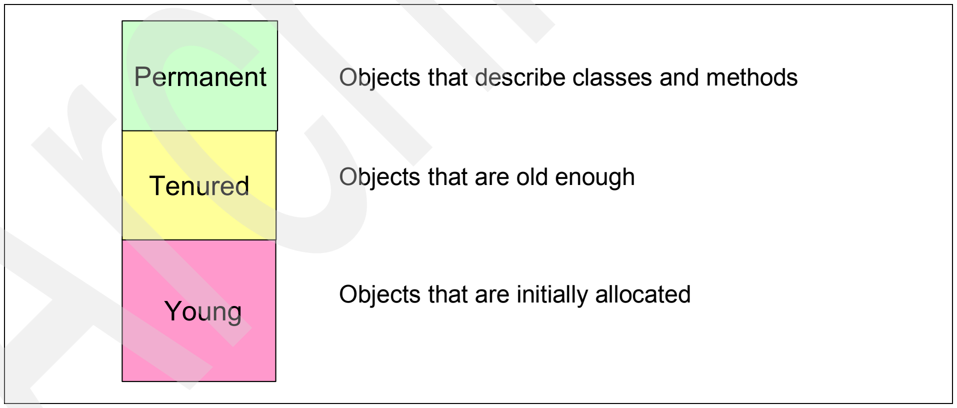

Permanent
Objects that describe classes and methods
Tenured
Objects that are old enough
Young
Objects that are initially allocated

Figure 12-6 JVM heap generations

When objects are first created, they are allocated in the young generation space. As these objects age, they move to the tenured generation space. Objects that describe classes and methods are stored in the permanent generation space.

Minor collections, which run only against the young generation space, execute faster than collections that run against the entire heap. Ideally, minor collections involve the removal of many objects. *Major collections* run against the tenured generation space. They run less frequently and take longer than minor collections, and typically remove fewer objects. The size of the permanent generation space is important for applications that dynamically generate and load a large number of classes. If the permanent generation space does not have enough memory, then an OutOfMemoryError is thrown.

At JVM initialization, only the initial (minimum) size of the Java heap is allocated. The maximum size of the Java heap memory is virtually reserved but not allocated. The young, tenured, and permanent spaces each consist of allocated and virtual memory.

Generation sizing involves trading off throughput, wait time, memory usage, and memory availability. Figure 12-7 presents a decision matrix for the sizing of the young and tenured generation spaces.

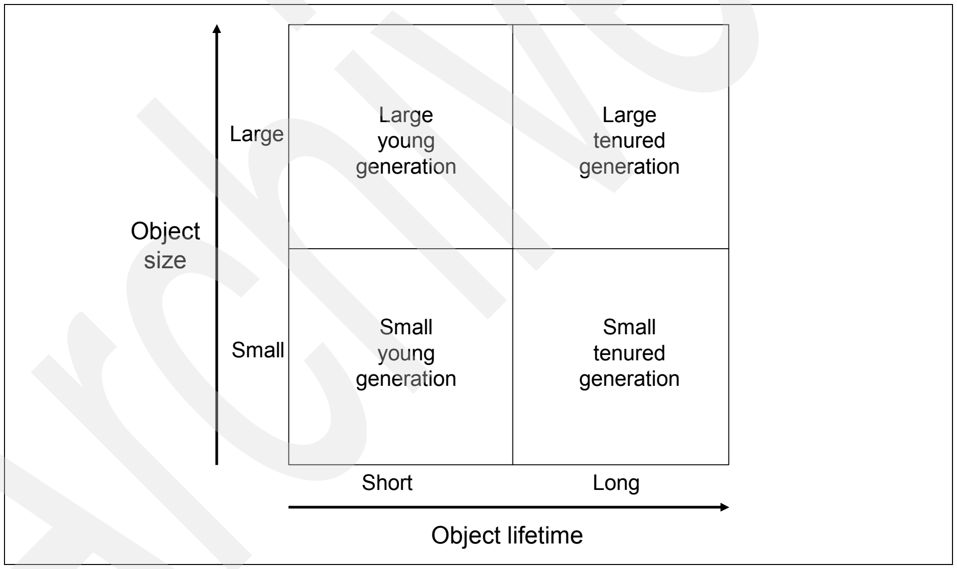


Figure 12-7 Heap generation sizes

Generation size is a function of the number of objects of specified size and lifetime. For example, consider a middle-tier application that serves documents to many concurrent users. Many documents are held at once in memory for transform and subsequent download to clients before being released. Since these documents are not reused, a large young generation is implied.

A batch job that retrieves and processes documents sequentially implies a small young generation. That is, only a few documents are processed for garbage collection each cycle, and the application executes at a steady pace. However, observe that specifying a larger than necessary young generation causes Garbage Collector to run less often but to run for a longer period. The application runs in short burst, which may be desirable if other applications can use the system effectively during the time that your application is not performing garbage collection.

Acceptable heap generation sizes, garbage collection frequencies, and Garbage Collector run times all depend on the memory usage pattern of your application. When the virtual machine's default configuration is inadequate, you can tune JVM memory management by using the command line options that we describe in the following section.

12.6 Startup parameters

Most JVM implementations have optional startup (command line) parameters that are used to adjust Java memory allocation and garbage collection. In the following sections, we discuss some of the most popular and commonly supported parameters. Not all JVM implementations support all of these options.

12.6.1 Supported commands and available options

The commands presented in the following sections are used to determine which other commands are available for your particular version of the JVM.

-help

The **`java -help`** or **`java -?`** command produces a list of the supported command line options as shown in Example 12-1.

Example 12-1 The java -help command

```
C:\>java -help
Usage: java [-options] class [args...] (to execute a class)
   or  java [-options] -jar jarfile [args...] (to execute a JAR file)

where options include:
    -client       to select the "client" VM
    -server       to select the "server" VM
    -hotspot      is a synonym for the "client" VM  [deprecated]
                  The default VM is client.
    -cp <class search path of directories and zip/jar files>
    -classpath <class search path of directories and zip/jar files>
                  A ; separated list of directories, JAR archives,
                  and ZIP archives to search for class files.
    -D<name>=<value>
                  set a system property
    -verbose[:class|gc|jni]
                  enable verbose output
    -version      print product version and exit
    -version:<value>
                  require the specified version to run
    -showversion  print product version and continue
    -jre-restrict-search | -jre-no-restrict-search
                  include/exclude user private JREs in the version
search
    -? -help      print this help message
    -X            print help on non-standard options
    -ea[:<packagename>...|:<classname>]
    -enableassertions[:<packagename>...|:<classname>]
                  enable assertions
    -da[:<packagename>...|:<classname>]
    -disableassertions[:<packagename>...|:<classname>]
                  disable assertions
    -esa | -enablesystemassertions
                  enable system assertions
    -dsa | -disablesystemassertions
                  disable system assertions
    -agentlib:<libname>[=<options>]
                  load native agent library <libname>, e.g.
-agentlib:hprof
                    see also, -agentlib:jdwp=help and
-agentlib:hprof=help
    -agentpath:<pathname>[=<options>]
                  load native agent library by full pathname
    -javaagent:<jarpath>[=<options>]
                  load Java programming language agent, see
java.lang.instrument

    -splash:<imagepath>
                  show splash screen with specified image
```

-X

The **java -X** command displays the syntax of the non-standard options as shown in Example 12-2. The **-X** options are non-standard and subject to change without notice.

Example 12-2 The options of the java -X command

```
C:\Documents and Settings\Administrator>java -X
    -Xmixed           mixed mode execution (default)
    -Xint             interpreted mode execution only
    -Xbootclasspath:<directories and zip/jar files separated by ;>
                      set search path for bootstrap classes and 
resources
    -Xbootclasspath/a:<directories and zip/jar files separated by ;>
                      append to end of bootstrap class path
    -Xbootclasspath/p:<directories and zip/jar files separated by ;>
                      prepend in front of bootstrap class path
    -Xnoclassgc       disable class garbage collection
    -Xincgc           enable incremental garbage collection
    -Xloggc:<file>    log GC status to a file with time stamps
    -Xbatch           disable background compilation
    -Xms<size>        set initial Java heap size
    -Xmx<size>        set maximum Java heap size
    -Xss<size>        set java thread stack size
    -Xprof            output cpu profiling data
    -Xfuture          enable strictest checks, anticipating future 
default
    -Xrs              reduce use of OS signals by Java/VM (see 
documentation)
    -Xcheck:jni       perform additional checks for JNI functions
    -Xshare:off       do not attempt to use shared class data
    -Xshare:auto      use shared class data if possible (default)
    -Xshare:on        require using shared class data, otherwise fail.
```

12.6.2 Performance and analysis commands

The commands that are presented in the following sections are useful for diagnostic purposes.

-Xprof

The -Xprof option causes the JVM to report CPU activity. A report is generated for each terminating thread that shows the methods that consumed the most time while that thread ran.

Example 12-3 shows an example of output that is generated by using the `-Xprof` parameter in a batch program that is connected to an OnDemand server over dial-up. Each thread's output is delimited by a header that starts, "`Flat profile of...`" and provides these details:

- In the first column, the percentage of CPU time spent executing a given method (or function in the case of a JNI invocation)
- In the second column, the amount of time spent in Java code
- In the third column, the amount of time spent in native code
- In the fourth column, the name of the method or function that is being executed

The methods are ordered by percentage of CPU time taken to execute, with the method that took the longest listed first. The global summary at the end of the profile provides information such as how much time was spent loading classes.

Example 12-3 Sample -Xprof output run on Windows

```
Flat profile of 82.13 secs (6778 total ticks): Thread-0

  Interpreted + native   Method
 97.4%     0  +  6601    com.ibm.edms.od.ArsWWWInterface.apiLogon
  0.9%     0  +    58    com.ibm.edms.od.ArsWWWInterface.apiLogoff
  0.6%     0  +    40    com.ibm.edms.od.ArsWWWInterface.apiGetUserInfo
  0.5%     0  +    31    java.lang.ClassLoader$NativeLibrary.load
  0.4%     0  +    28
com.ibm.edms.od.ArsWWWInterface.apiInitializeNative
  0.1%     0  +     4    java.io.FileOutputStream.writeBytes
  0.0%     0  +     3    java.io.FileOutputStream.open
  0.0%     0  +     3    java.io.WinNTFileSystem.getBooleanAttributes
  0.0%     0  +     1    java.lang.ClassLoader.defineClass1
  0.0%     0  +     1    java.util.zip.ZipFile.read
  0.0%     0  +     1    java.io.FileOutputStream.close0
  0.0%     0  +     1    java.lang.Thread.sleep
  0.0%     1  +     0    java.util.Arrays.copyOf
  0.0%     0  +     1    TThread.run
 99.9%     1  +  6773    Total interpreted

  Thread-local ticks:
  0.0%     2             Class loader
  0.0%     2             Unknown: thread_state

Flat profile of 82.84 secs (6838 total ticks): main

  Interpreted + native   Method
 12.3%     0  +     7    java.io.FileOutputStream.writeBytes
  8.8%     0  +     5    java.io.WinNTFileSystem.getBooleanAttributes
  8.8%     0  +     5    java.io.WinNTFileSystem.getLength
  7.0%     0  +     4    java.util.zip.ZipFile.open
  3.5%     0  +     2    java.lang.Thread.sleep
  3.5%     0  +     2    java.util.zip.ZipFile.read
  3.5%     0  +     2    java.util.TimeZone.getSystemTimeZoneID
  3.5%     0  +     2    java.io.FileInputStream.readBytes
  1.8%     0  +     1    java.io.FileInputStream.open
  1.8%     1  +     0    java.lang.Math.min
  1.8%     1  +     0    java.util.regex.Pattern.peek
  1.8%     1  +     0    sun.misc.URLClassPath$JarLoader.getJarFile
  1.8%     1  +     0    java.util.ResourceBundle.getBundleImpl
  1.8%     1  +     0    java.io.BufferedReader.readLine
  1.8%     1  +     0    java.lang.String.toLowerCase
  1.8%     1  +     0    sun.misc.FloatingDecimal.<clinit>
  1.8%     1  +     0    Main.main
 66.7%     8  +    30    Total interpreted

  Thread-local ticks:
 99.2%  6781             Blocked (of total)
 33.3%    19             Class loader

Flat profile of 6.43 secs (4 total ticks): DestroyJavaVM

  Interpreted + native   Method
 25.0%     1  +     0    java.util.IdentityHashMap.keySet
 25.0%     1  +     0    Total interpreted

  Thread-local ticks:
 50.0%     2             Class loader
 25.0%     1             Unknown: thread_state

Global summary of 89.30 seconds:
100.0%  6844             Received ticks
  0.0%     1             Compilation
  0.3%    23             Class loader
  0.0%     3             Unknown code
```

-verbose[:class|gc|jni]

The `-verbose:gc` option causes the JVM to output information about each garbage collection event. In Example 12-4, you see two minor garbage collections and a full garbage collection.

- The number before the arrow shows the total size of all objects before the garbage collection.
- The number after the arrow gives the total size of all live objects after the garbage collection.
- The number in parenthesis gives the total available heap space in the young and tenured generation spaces. The permanent generation is not counted.

The value `0.0042673 secs` is the amount of time that it took the Garbage Collector to run.

Example 12-4 Sample output from the -verbose:gc parameter

```
[GC 3938K->3918K(5056K), 0.0042673 secs]
[GC 4814K->4791K(5696K), 0.0043874 secs]
[Full GC 4791K->2433K(5696K), 0.0370195 secs]
```

-XX:+PrintGCDetails

The `-XX:+PrintGCDetails` option causes the JVM to output detailed information about each GC event. As illustrated in Example 12-5, this information includes:

- The generation or generations that are affected
- Space allocated to objects before and after garbage collection
- Total available space
- Time taken to perform the garbage collection

After all threads terminate, a heap summary is produced that shows the memory usage for the overall process.

Example 12-5 Sample output by using the -XX:+PrintGCDetails option

```
[GC [DefNew: 896K->64K(960K), 0.0191265 secs] 896K->245K(5056K),
0.0192538 secs] [Times: user=0.02 sys=0.00, real=0.02 secs]
[GC [DefNew: 960K->64K(960K), 0.0041025 secs][Tenured:
4456K->1401K(4480K), 0.0278220 secs] 4531K->1401K(5440K), 0.0324628
secs] [Times: user=0.03 sys=0.00, real=0.03 secs]
[GC [DefNew: 896K->896K(960K), 0.0000514 secs][Tenured:
4384K->4639K(5120K), 0.0414162 secs] 5280K->4639K(6080K), 0.0418810
secs] [Times: user=0.04 sys=0.00, real=0.04 secs]
Heap
def new generation   total 960K, used 599K [0x26390000, 0x26490000,
0x26490000)
```

```
eden space 896K,  66% used [0x26390000, 0x26425e10, 0x26470000)
from space 64K,   0% used [0x26480000, 0x26480000, 0x26490000)
to   space 64K,   0% used [0x26470000, 0x26470000, 0x26480000)
tenured generation   total 5120K, used 4138K [0x26490000, 0x26990000,
0x26990000)
the space 5120K,  80% used [0x26490000, 0x2689aba8, 0x2689ac00,
0x26990000)
compacting perm gen  total 12288K, used 295K [0x26990000, 0x27590000,
0x2a990000)
the space 12288K,   2% used [0x26990000, 0x269d9ed8, 0x269da000,
0x27590000)
ro space 8192K,  62% used [0x2a990000, 0x2ae92a28, 0x2ae92c00,
0x2b190000)
rw space 12288K,  52% used [0x2b190000, 0x2b7d86b8, 0x2b7d8800,
0x2bd90000)
```

–XX:+PrintGCTimeStamps

The `–XX:+PrintGCTimeStamps` option when used with the `-XX:+PrintGCDetails` option produces a time stamp at the start of each garbage collection in addition to the information provided by `-XX:+PrintGCDetails`. The time stamps are relative to the start of process execution and reveal the timing and duration of garbage collection events. Example 12-6 shows sample output is produced by using this option.

Example 12-6 Sample output by using -XX:+PrintGCTimeStamps

```
228.657: [GC 228.657: [DefNew: 896K->64K(960K), 0.0184965 secs]
896K->245K(5056K), 0.0188488 secs] [Times: user=0.02 sys=0.00,
real=0.02 secs]
235.151: [GC 235.151: [DefNew: 960K->64K(960K), 0.0154360 secs]
1141K->1121K(5056K), 0.0160249 secs] [Times: user=0.01 sys=0.00,
real=0.01 secs]
241.684: [GC 241.684: [DefNew: 960K->64K(960K), 0.0075152 secs]
2017K->2008K(5056K), 0.0077535 secs] [Times: user=0.01 sys=0.00,
real=0.01 secs]
```

12.6.3 Memory allocation commands

When the JVM is initialized the maximum heap space defined by `-Xmx` is virtually allocated to the JVM. The `-Xms` parameter defines the amount of memory that is physically allocated. As the JVM requires more memory, it is allocated up to the maximum size determined by `-Xmx`. The JVM also allocates memory for other things, including a stack for each thread. It is normal for the total memory consumed by the JVM to exceed the value of `-Xmx`.

-Xms

The `-Xms` parameter changes the initial (minimum) size of the Java heap memory allocation from its default value. The default value is listed in Table 12-1 on page 230. The value is specified in bytes. Typical values are in the MB range. The following examples show the different syntax for specifying memory:

- `-Xms6291456`
- `-Xms6144k`
- `-Xms6m`

-Xmx

The `-Xmx` parameter changes the maximum size of the Java heap memory allocation from its default value. The default value is listed in Table 12-1 on page 230. The value is specified in bytes. Typical values are in the MB range. The following examples show the different syntax for specifying memory:

- `-Xmx83886080`
- `-Xmx81920k`
- `-Xmx80m`

Heap allocation: Sometimes it is advantageous to allocate the entire heap at startup, which reduces the number of memory allocations and provides the largest amount of contigous memory. To do this, set the minimum heap size specified by `-Xms` and maximum heap size specified by `-Xmx` to the same value.

-Xmn

The `-Xmn` parameter indicates the size of the heap for the young generation space.

Value: The value of `-Xmn` should be less than the value of `-Xmx`.

-Xss

Each Java thread has two stacks, one for Java code and one for native code. The `-Xss` option sets the maximum thread stack size that can be used by the native code in a thread. Every thread that is spawned during the execution of the program has the specified value as its native stack size.

`-Xss` is specified in bytes. The default stack size is 512 KB (`-Xss512k`).

-XX:MinHeapFreeRatio and -XX:MaxHeapFreeRatio

For the IBM JVM, the options are `-Xminf<0-1>` and `-Xmaxf<0-1>`. For the Sun JVM, they are `-XX:MinFreeHeapRatio=<0-100>` and `-XXMaxHeapFreeRatio=<0-100>`. Use the **`Java -X`** command to verify the parameters for your JVM.

When a garbage collection occurs, the JVM resizes the heap to keep its usage within the range specified by `-Xms` and `-Xmx`. When possible, the new size reflects a desired fraction of free space, determined by the following parameters:

```
-XX:MinHeapFreeRatio=<minimum>
-XX:MaxHeapFreeRatio=<maximum>
```

Memory management involves trading off Garbage Collector run time and run frequency. If Garbage Collector runs often while always freeing memory, the Java heap is probably filling too quickly, and the application can benefit from a larger heap. In addition to using the `-Xmx` option to specify a larger heap, you can adjust the heap free ratio so that the every current heap allocation anticipates your application's forward needs.

For example, if you specify a minimum free space ratio of 30% (expressed as 0.3 for the IBM JVM or 30 for the Sun JVM), then the virtual memory expands the heap to try to keep at least 30% of the heap free. By increasing this number beyond the default setting, you force the virtual memory to increase the heap size in anticipation of increasing usage.

Setting the initial memory allocations `-Xmx` and `-Xms` to the same size prevents dynamic resizing of the heap, making any FreeRatio setting superfluous.

12.6.4 Customizing the heap size

The heap size significantly impacts garbage collection, and hence, the performance of Java programs. To avoid heap fragmentation and to prevent out-of-memory conditions, for most cases, start your tuning process with the maximum Java heap size parameter (`-Xmx`) set to 256M and then adjust up or down as needed.

Even if the minimum (`-Xms`) heap is small, you must specify the maximum heap size to ensure that one contiguous area is available should the heap require expansion.

To evaluate virtual memory management:

1. Use a load generator program to test the performance of the Web server or application under the maximum expected load.
2. Use the `-verbosegc` option to measure time and resources that are consumed by the garbage collection.
3. Redirect standard error and standard output to a log file as in the following example:

   ```
   Java <parameters> -verbosegc >> logfile.txt
   ```
4. Review the timestamps of the log outputs. Note the frequency of garbage collections and the time taken for each. Also note the average memory usage by checking the heap size after each garbage collection.

Garbage Collector should run in milliseconds, and a full garbage collection should never exceed several seconds.

If the garbage collection process always frees most of your heap and the process runs for too long a period, consider reducing the heap size so that Garbage Collector runs more often. Always remember to retest your application under maximum load to verify the impact of changes.

If your system spends too much time on garbage collection, it is possible that your heap size is larger than the available physical memory, causing the operating system to swap portions of the heap from memory to disk and back. Your system administrator can help you determine the amount of free physical memory on your system.

Tip: Setting `-Xms` and `-Xmx` to the same value can improve performance by eliminating the memory allocation and deallocation that are involved in keeping the heap size within a range.

Attention: Monitor your memory usage. Increase available memory as you increase the number of processors.

12.6.5 Customizing the object generations

The relative size of the young generation space determines the ratio of minor collections to full collections. For a given heap size, the larger the young generation, the less often minor collections occur. A larger young generation space implies a smaller tenured generation space, which increases the frequency of major collections. The optimal young generation space to tenured generation space ratio depends on the distribution of lifetimes for the objects that are created by your application.

Young generation space size can be adjusted by using the `NewRatio` parameter. For example, setting `-XX:NewRatio=3` means that the ratio between the young generation space and the tenured generation space is one to three (1:3). That is, the size of the young generation is one fourth of the total heap.

The `NewSize` and `MaxNewSize` parameters control the minimum and maximum size of the young generation space. Setting parameters equal to one another has the same effect on the young generation space as setting the `-Xms` and `-Xmx` parameters equal on the total heap.

The design and usage of your ODWEK application determines the ratio of the size of the young generation space to size of the tenured generation space. Enough free memory must be reserved in the tenured generation space to accommodate all the live objects from the young generation space.

Tip: If your application creates large numbers of objects whose life times are skewed heavily toward either transience or long life, consider tuning the JVM's heap generations.

12.6.6 ODWEK Java API memory usage

Most instances of native heap exhaustion occur while retrieving documents because document retrieval is the most memory intensive ODWEK operation. At least three components within your ODWEK-based application consume memory:

- The JNI (native) portion of the ODWEK Java APIs
- The Java component of the ODWEK Java APIs
- The Java component of the application

Requests for data from the OnDemand server flow from your application through the Java component of the ODWEK Java APIs to the native component. The native component makes the TCP/IP connection to the OnDemand server, sends requests, and receives the results. Upon receipt of the results, bulk data, such as Advanced Function Presentation (AFP) resources and document data, are stored in the native heap.

Adjust the size of the native heap, by allocating a smaller maximum Java heap to free more native memory or vice versa, according to the size of your documents and the expected number of concurrent retrievals. This adjustment entails a kind of balancing act, since the Java component of the ODWEK Java APIs caches document data in Java arrays. The size of the Java heap must be large enough to accommodate the data, but in general, memory allocation should be biased in favor of native space.

The sizes of the Java and native stacks, which reflect the number and depth of active method calls, are most affected by the number of concurrent threads that are maintained by the Web server or application. The number of threads usually correlates with the number of concurrent users.

Tip: Consider the following useful guideline for allocating memory:

```
Native heap size = 2 x (average uncompressed document size) x
(average number of concurrent document requests)

Java heap size = (average document size) x (average number of
concurrent document requests)
```

12.7 Other performance areas

Web server and stand-alone applications that use the ODWEK Java APIs consume considerable system resources. The anticipated operating mode of ODWEK applications is to serve large numbers of users (tens of thousands) by extracting large quantities of data (documents) from large archives (spanning multiple years). Data is preprocessed if needed, converting AFP to HTML, for example, and presented to users with fast response times (in seconds).

Accomplishing this processing requires planning adequate system capacity and ensuring the best use of available resources. In the following sections, we look briefly at subsystems that can become bottlenecks and that can be tuned to improve performance. Monitoring and tuning procedures differ by platform. Therefore, the discussion is intended mainly to direct you to areas of interest.

12.7.1 Network

Web server applications establish multiple network connections to the OnDemand server and to browsers or other clients. Ensure that TCP/IP stacks and network routes to both are configured for efficiency. Terminology and support for parameters given in the following sections vary by platform. Therefore, we present general descriptions, rather than specific instructions.

Number of sockets

Web server applications connect to the OnDemand server and browser clients, and constantly create and destroy sockets. Verify that your system defines enough sockets and that your application is not delayed by waiting for in-use sockets to be freed.

TCP/IP buffer sizes

Default buffer sizes on most systems are usually smaller than the documents that are transferred. Set the buffer sizes to "slightly larger" than the size of the document that is being downloaded or to the largest possible size that is less than that. The TCP/IP buffer size minimizes flow control overhead and the number of TCP/IP sends and receives per document.

For example, if the TCP/IP receive buffer is too small, the receive window is prone to overrun. Upon overrun, the flow control mechanism stops data transfer until the receive buffer is empty. Flow control can consume significant CPU time and result in additional network latency.

Keep in mind that TCP/IP buffers can be too large. If applications do not process data fast enough, paging can increase. The goal is to specify a value that is large enough to avoid undue flow control overhead, but not so large that the buffer accumulates more data than the system can process.

Attention: Be extremely careful when considering changing the TCP/IP buffer size. It cannot be done on a per-application basis. Setting it with an improper value might cause problems to the entire network traffic.

nodelayack option

If the `nodelayack` option is left unset, TCP waits for the buffer to fill before sending. This option reduces the number of TCP sends and thus decreases stack overhead. It is good for sending large amounts of data (documents). When small quantities of data are to be sent, as for requests, you normally want immediate transmission for improved response time and accept additional CPU utilization as a cost for the additional flow control.

Setting the `nodelayack` option causes the data to be sent by TCP/IP as soon as applications request that it be sent. The combination of a large buffer size and setting the `nodelayack` option usually improves overall performance for mixed small (control) and large (data) transmissions.

Max user port (32768)

The Max user port parameter, when available, determines the highest port number (and thus the total number of ports) that TCP/IP can assign when an application requests a port from the system. See 13.2.3, "Performance degrades with a large number of server connections; the OnDemand server refuses network connections (Windows only)" on page 262.

MaxConnect Backlog

The MaxConnect Backlog parameter indicates the maximum number of connections that can be queued when waiting for TCP socket assignment. Increase this number if many connection attempts are received simultaneously and you know that they will be serviced relatively quickly before the waiting connections time out.

Keepalive_interval

The `Keepalive_interval` option is the default TCP keep-alive interval for applications that enable the `SO_KEEPALIVE` socket option. Do not override the interval by using the `TCP_KEEPALIVE` option. A typical range is 0-35791 minutes, with a default of 120. A value of zero (`0`) disables the keep-alive function so that sockets for which `SO_KEEPALIVE` is specified do not perform the TCP keep-alive function. In this case, sockets that specify an interval by using `TCP_KEEPALIVE` continue to send keep-alive probes.

FINWAIT2TIME (also TcpTimedWaitDelay)

FINWAIT2TIME is the number of seconds that a TCP connection should remain alive waiting for more data to be sent. The default on some systems is five minutes, which means that many sockets that are no longer in use by applications remain unavailable for reuse for a period of five minutes.

When thousands of sockets are constantly created and disposed, the delay is an unnecessary load on the system. Performance testing usually reveals an appropriate value of considerably less than five minutes.

MTU_size

The maximum transmission unit (MTU) in bytes. The MTU_size value can be up to 65535. The default value of this field can be as low as 576. Larger units are more efficient for transmitting large amounts of data, but are more costly to recover in the event of a transmission error. In general, a smaller MTU works better on slow or unreliable networks that drop packets frequently, such as over dial-up or DSL. Larger MTU sizes are appropriate for fast, reliable networks.

Network adapters

Your Web server application transmits a lot of data both from the OnDemand server and to browser clients. Verify that your network adapters are running both full duplex and at the maximum transfer rate that is supported by the cards and your network.

12.7.2 Disk

Disk speed and I/O buffering have a dramatic effect on the performance of Web server applications that heavily depend on disk access, database support, and extensive messaging. Disk input or output subsystems that are optimized for performance, for example Redundant Array of Independent Disks (RAID) arrays, high-speed drives, and dedicated caches, are essential contributors to application server performance in these environments. Application servers with smaller disk requirements can benefit from a mirrored disk drive configuration that improves reliability and has good performance.

Spread the disk processing across as many disks as possible to avoid contention issues that can occur with one- or two-disk systems. Placing database tables on separate disks from those that store database logs reduces contention and improves throughput.

12.7.3 Processor

In the absence of other bottlenecks, increasing the processor speed improves throughput and response time. A processor with a larger L2 or L3 cache yields higher throughput than a processor of the same speed with a smaller cache.

Highly multi-threaded applications benefit from systems with multiple processors because such systems can run multiple threads simultaneously. Many single-threaded applications can be refactored to spawn multiple threads to distribute load across multiple processors.

Eliminating unneeded server processes and increasing the amount of processor time (process priority) associated with the Web server or ODWEK application can also improve performance.

12.7.4 Physical memory

Increasing memory sufficiently to prevent the operating system from paging the virtual memory of processes to disk improves performance. Configure a minimum of 256 MB of memory for each processor and at least 512 MB per application server instance. Adjust the available memory when the system pages and the processor utilization is low because of the paging. Memory access speed often depends on the number and placement of the memory modules. Check the hardware manual to ensure that your configuration makes the best use of installed memory.

13

Troubleshooting

In this chapter, we describe resources and methods for troubleshooting OnDemand Web Enablement Kit (ODWEK)-based Java applications.

We discuss the following topics in this chapter:

- ODWEK error reporting and trace logging
- Common problems and their solutions
- Java dump (javacore)
- Other Java diagnostic tools
- Testing tools
- Getting support

13.1 ODWEK error reporting and trace logging

The ODWEK reports runtime error conditions by throwing an ODException and, if trace logging is enabled, by writing log entries to the arswww.trace file. The trace file can be configured to capture considerable detail and is useful for performance tuning and general troubleshooting.

13.1.1 ODException class

The ODException class extends Java's Exception class and handles all exceptions that are thrown by the ODWEK Java APIs. The exceptions that are thrown by the ODWEK Java APIs should be handled as with any other Java exception. Exceptions are queried for an error code by calling the getErrorId() method. Obtain any additional detail message by calling the getErrorMsg() method.

Tip: For information about interpreting Content Manager OnDemand (OnDemand) error codes, see *IBM Content Manager OnDemand: Messages and Codes*, SC27-1379, which contains detailed descriptions. Although the guide lists the error codes with the server-specific prefix ("ARS") as used in the OnDemand System Log, the integer maps correctly to the error ID that is obtained from an ODException class.

13.1.2 Trace logging

The arswww.trace file contains time-stamped debugging information that is generated by the ODWEK Java APIs. The file contains entries from both the Java and native components. A trace file is helpful for debugging your application and for identifying performance bottlenecks.

Enabling the ODWEK trace engine

To enable trace logging, specify the appropriate traceDir and traceLevel parameter values when initializing the ODConfig object that is passed to new ODServer connections. See Example 13-1.

Example 13-1 Specifying the trace setting when initializing ODConfig

```
public ODConfig (String afpViewer,
                String lineViewer,
                String metaViewer,
                long maxHits,
                String appletDir,
                String language,
                String tempDir,
                String traceDir,
                int traceLevel)
```

The traceLevel parameter controls the level of detail and minimum severity of entries that is written to the arswww.trace file. Table 13-1 describes the output that each trace level produces.

Table 13-1 Trace level description

Trace level	Description
0 (default)	No logging is enabled.
1	Log only ERROR events.
2	Log only ERROR and WARNING events.
3	Log ERROR, WARNING, and INFO events.
4	Log ALL events.

Higher levels are used to troubleshoot current ODWEK issues. Lower levels are used to monitor an ODWEK application that is in a steady state. For example, in a test environment, setting traceLevel to 1 entails minimal overhead while informing you of errors.

Tip: When diagnosing problems, set traceLevel to 4. At level 4, all error details are written to the trace file. Error traces often relate information that cannot be obtained from corresponding ODExceptions.

Important: A full trace (level 4) can degrade performance. Always set the trace level to 1 (ERROR) so that ERROR generation occurs when there is an error. In addition, always monitor the trace files to catch and respond to errors as soon as they occur. The ERROR trace level has little or no impact in production, as long as errors are not occuring.

Trace files are located in the directory given by the traceDir parameter. A new trace file is created when an application's first ODServer object is initialized. A new trace file is also created when the first new ODServer is initialized after all previous ODServer connections are logged-off and terminated. Any existing trace file is renamed to `arswww.trace<timestamp>`. Ensure that adequate disk space is available on the directory's file system. The historical trace files must be deleted manually to avoid exhausting the file system.

13.1.3 Analyzing the trace file

As mentioned, the trace file contains in-depth details of thrown exceptions. At its most detailed level (level 4), the trace log also records the flow of execution through the API, including calls to native modules, which can help diagnose OnDemand server problems.

The subsecond granularity of trace time stamps is useful for debugging performance problems and unresponsive applications. Timestamp information can be used to help isolate bottlenecks in your application, the ODWEK Java API, or OnDemand server request processing. You can see exactly which methods are in use by your program and the time that is spent by executing each method.

The output of the arswww.trace file includes the following columns, which are output in a comma-separated format (CSV):

- PPID: The current working or child process ID
- PID: The parent process ID
- TID: The current thread ID for this request (typically maps to a user session)
- DATE: Date in the month/day/year format
- TIME: Transaction time stamp
- LEVEL: Level of message
- FUNCTION: The native function to which the message applies
- OUTPUT: Detailed message text

If you receive an ENTER or RETURN message, then a numeric value that represents the ODWEK session ID for the function call is in the OUTPUT field.

The messages can have one of the following possible levels:

- ERROR: Error messages
- WARNING: Warning messages
- INFO: Informational messages that assist problem determination
- ENTER: Function flow messaging
- RETURN: Function flow messaging

Java_com_ibm_edms_od_: Function names with Java_com_ibm_edms_od_ prepended indicate the base Java Native Interface (JNI) call that is made from the Java virtual machine (JVM). All other calls are internal ODWEK library calls.

Error messages

To check for error messages, search the LEVEL column for ERROR. Examine the FUNCTION column and determine which function was executing when the error occurred. Then work your way backward through the trace to determine the cause of the error, which may be the result of the following common culprits:

- Bad data passed to the function
- Network failure
- Poor response time from the server due to network bottlenecks or server overload

  Check the server log to determine the server's load.
- Out-of-memory condition

Errors that are thrown by Content Manager OnDemand native components are written as text in the patterns `CSV_RC=<num>` and `CSV_ID=<Num>`. A detail message is normally provided. See 13.1.5, "Return codes and message IDs" on page 258, for tables of return codes and message IDs.

Performance

To check for performance issues, compare consecutive time stamps in the TIME column. Time differences between the most consecutive entries are subsecond except while data is retrieved from the server. Identify any large time difference between one time stamp and the next and note the function that is indicated in the FUNCTION column.

The time that is required to return data from the server might be several seconds or more. Retrieval performance depends on the OnDemand server's capacity, the quantity of data that is returned, network speed, and (if present) the middle tier server's capacity. It is helpful to determine normal response times for your systems during off-peak access as a benchmark to assess performance problems later.

13.1.4 Trace log sample

The raw arswww.trace output for a typical ODHit.retrieve() call with traceLevel set to 4 is illustrated in Example 13-2. This example shows a single PPDID and TID. In real life, most traces show multiple TIDs, or rather multiple threads, and the entries of other threads must be filtered out.

Example 13-2 Snippet from the ODHit.retrieve() call with traceLevel=4

```
4196,3272,316,12/17/07,13:59:42.145,INFO,Java_com_ibm_edms_od_ArsWWWInt
erface_apiRetrieve,DocID =
v7126-19459-19460-19461-ACF1-1FAAA-0-27629619-0-27629619-
85-68-0-1-0-^^ATEST TIFF
4196,3272,316,12/17/07,13:59:42.145,ENTER,apiP_OpenFolderByName,
4196,3272,316,12/17/07,13:59:42.145,INFO,apiP_OpenFolderByName,Opening
[test-LargeTiff]
4196,3272,316,12/17/07,13:59:42.145,RETURN,apiP_OpenFolderByName,RC=0
4196,3272,316,12/17/07,13:59:42.145,INFO,Java_com_ibm_edms_od_ArsWWWInt
erface_apiRetrieve,Doc Type is 'T'
4196,3272,316,12/17/07,13:59:42.145,INFO,Java_com_ibm_edms_od_ArsWWWInt
erface_apiRetrieve,pViewer = native
4196,3272,316,12/17/07,13:59:42.145,ENTER,Util_updateSessionViewOpts,
4196,3272,316,12/17/07,13:59:42.145,RETURN,Util_updateSessionViewOpts,
4196,3272,316,12/17/07,13:59:42.145,INFO,Java_com_ibm_edms_od_ArsWWWInt
erface_apiRetrieve,Retrieve Data Compressed.
4196,3272,316,12/17/07,13:59:44.999,ENTER,JNIDataCallback,
4196,3272,316,12/17/07,13:59:46.181,RETURN,JNIDataCallback,RC=0
4196,3272,316,12/17/07,13:59:46.191,ENTER,JNIDataCallback,
4196,3272,316,12/17/07,13:59:46.191,RETURN,JNIDataCallback,RC=0
4196,3272,316,12/17/07,13:59:46.291,INFO,Java_com_ibm_edms_od_ArsWWWInt
erface_apiRetrieve,Document segment written to file [27629619] bytes
4196,3272,316,12/17/07,13:59:46.301,INFO,Java_com_ibm_edms_od_ArsWWWInt
erface_apiRetrieve,Document Compression Type is 'D'
4196,3272,316,12/17/07,13:59:46.301,INFO,Java_com_ibm_edms_od_ArsWWWInt
erface_apiRetrieve,DocType = T
4196,3272,316,12/17/07,13:59:46.301,INFO,Java_com_ibm_edms_od_ArsWWWInt
erface_apiRetrieve,AFP Viewer = 8
4196,3272,316,12/17/07,13:59:46.751,RETURN,Java_com_ibm_edms_od_ArsWWWI
nterface_apiRetrieve,1167736032 RC=0
```

Table 13-2 shows the arswww.trace details for the same retrieve() call that is given in Example 13-2. To obtain a similar table for your trace, import your file into Microsoft Excel® as CSV.

Table 13-2 Details of the arswww.trace file for the retrieve() call in Example 13-2

PPID	PID	TID	DATE	TIME	LEVEL	FUNCTION	OUTPUT
3272	4196	316	2/17/2007	3:59:42	ENTER	Java_com_ibm_edms_od_ArsWWWInterface_apiRetrieve	1167736032
3272	4196	316	2/17/2007	3:59:42	INFO	Java_com_ibm_edms_od_ArsWWWInterface_apiRetrieve	DocID = v7126-19459-19460-19461-ACF1-1FAAA-0-27629619-0-27629619-85-68-0-1-0-^TEST TIFF
3272	4196	316	2/17/2007	3:59:42	ENTER	apiP_OpenFolderByName	
3272	4196	316	2/17/2007	3:59:42	INFO	apiP_OpenFolderByName	Opening [test-LargeTiff]
3272	4196	316	2/17/2007	3:59:42	RETURN	apiP_OpenFolderByName	RC=0
3272	4196	316	2/17/2007	3:59:42	INFO	Java_com_ibm_edms_od_ArsWWWInterface_apiRetrieve	Document type is 'T'
3272	4196	316	2/17/2007	3:59:42	INFO	Java_com_ibm_edms_od_ArsWWWInterface_apiRetrieve	Viewer = native
3272	4196	316	2/17/2007	3:59:42	ENTER	Util_updateSessionViewOpts	
3272	4196	316	2/17/2007	3:59:42	RETURN	Util_updateSessionViewOpts	
3272	4196	316	2/17/2007	3:59:42	INFO	Java_com_ibm_edms_od_ArsWWWInterface_apiRetrieve	Retrieve data compressed.
3272	4196	316	2/17/2007	3:59:45	ENTER	JNIDataCallback	
3272	4196	316	2/17/2007	3:59:46	RETURN	JNIDataCallback	RC=0
3272	4196	316	2/17/2007	3:59:46	ENTER	JNIDataCallback	
3272	4196	316	2/17/2007	3:59:46	RETURN	JNIDataCallback	RC=0
3272	4196	316	2/17/2007	3:59:46	INFO	Java_com_ibm_edms_od_ArsWWWInterface_apiRetrieve	Document segment written to file [27629619] bytes

PPID	PID	TID	DATE	TIME	LEVEL	FUNCTION	OUTPUT
3272	4196	316	2/17/2007	3:59:46	INFO	Java_com_ibm_edms_od_ArsWWWInterface_apiRetrieve	Document compression type is 'D'
3272	4196	316	2/17/2007	3:59:46	INFO	Java_com_ibm_edms_od_ArsWWWInterface_apiRetrieve	DocType = T
3272	4196	316	2/17/2007	3:59:46	INFO	Java_com_ibm_edms_od_ArsWWWInterface_apiRetrieve	AFP Viewer = 8
3272	4196	316	2/17/2007	3:59:47	RETURN	Java_com_ibm_edms_od_ArsWWWInterface_apiRetrieve	167736032 RC=0

13.1.5 Return codes and message IDs

Errors that are thrown by Content Manager OnDemand native components are written as trace file text in the patterns `CSV_RC=<num>` and `CSV_ID=<Num>`. A detail message is normally provided. Table 13-3 gives the CSV_RC (return code) mappings in ODWEK. The error message is typically returned as MSGID=9, which indicates a miscellaneous error. In certain instances, it provides additional details for Level 3 support to provide more efficient problem determination.

Table 13-3 Return code mapping

CSV_RC	ERROR
0	CSV_RC_OKAY
1	CSV_RC_OKAY_WITH_MESSAGE
2	CSV_RC_CANCEL
3	CSV_RC_PASSWORD_CHANGE
4	CSV_RC_DOC_UNAVAILABLE
5	CSV_RC_INVALID_SEARCH
6	CSV_RC_NO_PERMISSION
7	CSV_RC_TIMEOUT
8	CSV_RC_DATA_CONVERSION_ERROR
9	CSV_RC_MISC_ERROR
10	CSV_RC_EXISTS

13.2 Common problems and their solutions

In this section, we cover the following common problems and their solutions:

- Application is unresponsive or JVM out-of-memory error occurs.
- Application terminates with a ‘DLL could not be found’ message.
- Performance degrades with a large number of server connections; the OnDemand server refuses network connections (Windows only).
- Document retrieval is long-running.
- Folder search does not produce a correct hitlist after upgrading ODWEK.
- Entire document is not retrieved.

13.2.1 Application is unresponsive or JVM out-of-memory error occurs

In the event of an application becomes unresponsive, you can analyze the Java dump. See 13.3, “Java dump (javacore)” on page 265.

When the JVM cannot allocate enough Java heap memory for the current operation, the JVM throws an out-of-memory error and exits.

Common on 32-bit platforms, memory allocation requires tuning when document sizes regularly exceed 100 MB. In most cases, the problem is caused by failure to allocate sufficient native memory to retrieve large documents, especially when several such documents are requested at once.

You might have to lower your maximum Java heap allocation to accommodate a larger native heap. See Chapter 12, “Memory and performance” on page 223, especially 12.6, “Startup parameters” on page 236, for more information about native and Java heap allocation. As a rule-of-thumb, set the native heap size based on the following equation:

```
2 x (average document size) x (number of concurrent users)
```

Applications that run on 32-bit Java platforms might continue to run out of memory despite allocation and tuning of the maximum possible 32-bit heap. Consider upgrading such 32-bit installations to 64-bit versions, which support a much larger address space for both native and Java heaps. A better solution is for them to modify their application to use the retrieve to file operation, which completely removes this problem.

When dealing with large AFP resources, also consider implementing custom AFP resource caching, by which you can avoid holding large AFP resources in memory, which is the default behavior of ODWEK. See 9.3, “AFP resource retrieval and custom caching” on page 185, for more information.

If the instability is traced to an operation other than document retrieval, see 13.1.3, “Analyzing the trace file” on page 254. If an application becomes unresponsive, but there is no javacore or JVM exit, see the following sections in this chapter:

- “Performance” on page 255
- 13.2.3, “Performance degrades with a large number of server connections; the OnDemand server refuses network connections (Windows only)” on page 262
- 13.2.4, “Document retrieval is long-running” on page 263

Tip: Try to recreate the problem by using a stand-alone or command-line test program. If the stand-alone test does not fail, verify that the failing application logs off and terminates all ODServer objects. Improperly handled connections are a *frequent cause* of resource problems. If documents have large AFP resources embedded, check whether ODWEK fails to allocate native memory for AFP resources. See 13.1.3, “Analyzing the trace file” on page 254.

13.2.2 Application terminates with a ‘DLL could not be found’ message

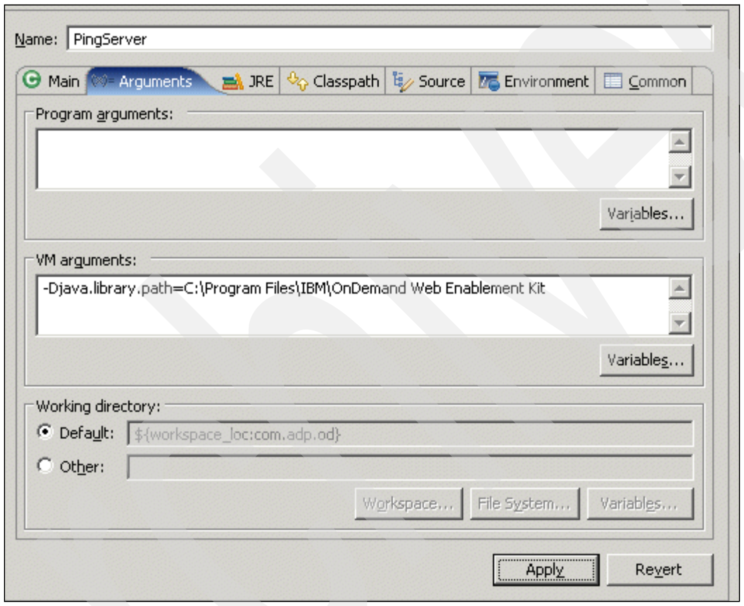
Application terminates with the message **“**`Dynamic Load Library 'ARS3WAPI32.DLL' could not be found`” or JVM throws an UnsatisfiedLinkError.

The ODWEK Java APIs are not 100% Pure Java™. There are both native and Java code. For the native code, the Java run time (JVM) must be told where to find these native libraries, just as it must be told where to find the Java classes that you want to execute. While Java classes are located by using the classpath setting (command line argument or environment setting), the location of native shared libraries is specified by the `java.library.path` virtual memory argument. This solves the loading of libraries that are loaded by using the System.loadLibrary Java method call. If any of these libraries have other libraries that they need to load, such as ICU, then these other libraries can only be found through the environment path variable.

To specify the `java.library.path` at the command line, use the -D parameter, as in the following example (Windows style, with ODWEK installed in the default directory):

```
java -cp ".;C:\Program Files\IBM\OnDemand Web Enablement
Kit\api\ODApi.jar" redbook.PingServer -Djava.library.path="C:\Program
Files\IBM\OnDemand Web Enablement Kit"
```

Users of Eclipse-based development environments (IDEs), including users of IBM Rational Application Developer and Rational Software Architect, can specify the `-D` argument in the Run window, as shown in Figure 13-1. To open the Run window, select **Run** → **Run**.



Name: PingServer

Program arguments:

VM arguments:
-Djava.library.path=C:\Program Files\IBM\OnDemand Web Enablement Kit

Working directory:
Default: ${workspace_loc:com.adp.od}
Other:

Figure 13-1 Run window in IBM Rational Software Architect 7.0

Configuration of user libraries or shared libraries for various environments is beyond the scope of this section. However a user-defined library is the preferred way to reference the ODWEK executable files. See 2.3, “Setting up a Web development environment by using Rational Application Developer” on page 43, for an Eclipse-based example.

13.2.3 Performance degrades with a large number of server connections; the OnDemand server refuses network connections (Windows only)

During periods of high usage, in which server connections are repeatedly opened and terminated, it is possible to exhaust the number of TCP/IP ports that are available for connections. You can check this by typing the following command at a command prompt:

```
netstat -an
```

Figure 13-2 shows how the result should look.

```
TCP 12.29.39.143:1445 12.29.39.153:4796 ESTABLISHED
TCP 12.29.39.143:1445 12.29.39.153:4845 ESTABLISHED
TCP 12.29.39.143:1445 12.29.39.153:4930 ESTABLISHED
TCP 12.29.39.143:4246 192.168.250.40:1504 TIME_WAIT
TCP 12.29.39.143:4247 192.168.250.40:1504 TIME_WAIT
TCP 12.29.39.143:4248 192.168.250.40:1504 TIME_WAIT
```

Figure 13-2 Sample netstat output

If the `netstat` reports usage of port numbers in the high 4000s, it is possible that the Windows default of 5000 user ports is exhausted by heavy traffic. When that happens, network connections to and from the affected machine are rejected.

To resolve this, update the registry values for TcpTimedWaitDelay and MaxerPort on machines that are subject to high network traffic.

TcpTimedWaitDelay

TcpTimed WaitDelay determines the time that must elapse before TCP/IP can release a closed connection and reuse its resources. This interval between closure and release is known as the *TIME_WAIT state* or *twice the maximum segment lifetime (2MSL) state*.

During this time, reopening the connection to the client and server costs less than establishing a new connection. By reducing the value of this entry, TCP/IP can release closed connections faster and provide more resources for new connections. Adjust this parameter if the running application requires rapid release, the creation of new connections, or an adjustment because of a low throughput caused by multiple connections in the TIME_WAIT state.

To view or set TcpTimedWaitDelay:

1. Type the **`regedit`** command.
2. Access the registry subkey HKEY_LOCAL_MACHINE\SYSTEM\ CurrentControlSet\ Services\TCPIP\Parameters, and create a new REG_DWORD value named `TcpTimedWaitDelay`.
3. Set the value to decimal 30, which is hexadecimal 0x0000001e. This sets the wait time to 30 seconds.
4. Stop and restart the system.

The default value is 0xF0, which sets the wait time to 240 seconds (4 minutes). We recommend that you use a minimum value of 0x1E, which sets the wait time to 30 seconds.

MaxUserPort

MaxUserPort determines the highest port number that TCP/IP can assign when an application requests an available user port from the system.

To view or set MaxUserPort:

1. Type the **`regedit`** command.
2. Access the registry subkey HKEY_LOCAL_MACHINE\SYSTEM\ CurrentControlSet\ Services\TCPIP\Parameters, and create a new REG_DWORD value named `MaxUserPort`.
3. Set this value to at least decimal 32768.
4. Stop and restart the system.

The default value is 0x1388 (5000 decimal) for most Windows versions. We recommend that you use a value from decimal 32768 to 65534. For more details, see the following Microsoft document:

`http://support.microsoft.com/kb/120642/`

13.2.4 Document retrieval is long-running

You can improve ODWEK performance and the user experience when retrieving large files by using one or more of the methods referenced in the following sections, depending on your application and data.

Implementing large object support

Implement large object support, so that users retrieve documents in segments as needed. See 9.4, “Segmented retrieval and large object support” on page 187.

Selecting an appropriate retrieval method

The ODWEK Java API getDocument() and getResource() methods perform faster than the more commonly-used retrieve() method if a conversion is applied. Select the fastest retrieval method that is compatible with the document's format and storage. See Chapter 9, "Document retrieval" on page 179.

Implementing custom AFP resource caching

Custom AFP resource caching can minimize the network transfer of large AFP resource data and prevent ODWEK from caching large but infrequently-used AFP resources in memory. See 9.3, "AFP resource retrieval and custom caching" on page 185.

Implementing alternative retrieval for large documents

Check the size of documents to be retrieved before retrieving them. For documents that exceed a threshold size, use a separate thread for retrieval (for example, retrieve to file versus memory) and call back to the client when the document is transferred. Make the retrieval operation cancellable and implement a transfer time-out as desired.

Canceling long-running requests

For any environment, regardless of its speed and sophistication, a document size and server load threshold exist beyond which clients must wait an impractically long time for document retrieval. This is especially relevant when requests are queued or when very large documents are transformed or otherwise processed on the server before streaming to the client. This threshold is lower for Web browsers because HTTP requests eventually time out.

ODWEK does not directly support query or data transfer timeouts. However an application can implement its own timeout feature and use the ODServer.cancel() method to end long-running server requests. Supported methods that can be canceled are logon, search, and retrieve. The call must be made in a separate thread. Providing the cancel functionality enables the application to respond to the client in a timely manner. Users can also find a Cancel button convenient. Providing Web clients with this feature helps to avoid orphaned server requests.

Optimizing the network architecture

The ODWEK Java APIs perform a great deal of client-server network communication. As such, increasing network speed and improving routing between distant clients and servers can improve the performance of ODWEK.

13.2.5 Folder search does not produce a correct hitlist after upgrading ODWEK

Because an existing ODFolder reference is now reused, open folders retain any previously set ODCriteria search values. Applications must either close folders immediately after use or set all of their criteria before each search, setting unused criteria values to `null`.

13.2.6 Entire document is not retrieved

If the issue cannot be ascribed to network problems, resource allocation failures, or other technical problems, check whether large object support has been enabled for the document's storage. Initial retrievals from large object-enabled sources return only the first segment of requested documents.

For more information about document retrieval with large object support enabled, see 9.4, "Segmented retrieval and large object support" on page 187.

13.3 Java dump (javacore)

When Java processes terminate or hang unexpectedly, the JVM creates a *Java dump file*. The file contains diagnostic information that describes the state of the operating system, the JVM, and the Java application at the time of the failure. For some JVM implementations, a Java dump is known as a *javacore*.

The contents of the dump file vary by operating system and JVM implementation, but generally provide information about threads, memory, the native stack, and locks. Some JVMs provide environment variables and runtime switches with which you can customize the Java dump.

Tip: Sometimes it is helpful to display a stack trace at various points in program execution. You can output the stack trace for the current thread by calling the Thread.dumpStack() method.

13.3.1 IBM Thread and Monitor Dump Analyzer

The IBM Thread and Monitor Dump Analyzer for Java Technology is a full function analyzer for JVM dumps. It is capable of analyzing one or more dump files and provides thread, heap and monitor diagnostic information.

For more information, see IBM Thread and Monitor Dump Analyzer in IBM alphaWorks® at the following address:

`http://www.alphaworks.ibm.com/tech/jca`

13.3.2 HeapAnalyzer

HeapAnalyzer is a graphical tool for discovering possible Java heap leaks. It analyzes a Java heap dump and produces a heap map and a list of heap leak suspect objects.

Find more information about HeapAnalyzer, see the following alphaWorks page:

`http://www.alphaworks.ibm.com/tech/heapanalyzer`

13.3.3 HeapRoots

HeapRoots is a another tool for debugging memory leaks in Java applications. This tool provides commands to analyze heap dumps. Use HeapRoots to investigate the heap object-by-object and to capture summary information and statistics about the heap and the objects that are stored.

For more information, see the following alphaWorks page:

`http://www.alphaworks.ibm.com/tech/heaproots`

13.4 Other Java diagnostic tools

All Java diagnostic tools are sensitive to the JVM level, platform, and operating system. Generated output and support for given features may vary. These diagnostic utilities use the JVM tools interface to obtain information about running Java processes and the status of the JVM.

13.4.1 The jmap command

The **jmap** command is a command-line utility that produces a memory map for a running JVM or Java application. We describe frequently used parameters in the sections that follow. The **jmap** command requires the following command-line usage:

```
jmap parameter pid
```

In this command, note the following explanation:

- `pid` indicates the process to be mapped.
- `parameter` is optional.

If no parameters are specified, then **jmap** prints a list of the shared objects that are loaded.

For more information, see the Sun Web page at the following address:

http://java.sun.com/j2se/1.5.0/docs/tooldocs/share/jmap.html

-histo parameter

The `–histo` parameter produces a histogram of all the classes in the heap. For each class in the heap, this parameter shows the class names, instance totals, and the total number of bytes that are consumed by those objects. Example 13-3 shows a sample of the output of **jmap -histo**.

Example 13-3 Sample snippet of jmap -histo output

```
C:\>jmap -histo 4024

num     #instances         #bytes  class name
----------------------------------------------
   1:          4016         663976  [Ljava.lang.Object;
   2:           223         209632  [I
   3:          1232         206976  com.ibm.edms.od.ODFolder
   4:          3078         184408  [C
   5:           423          86320  <constMethodKlass>
   6:          3099          74376  java.lang.String
   7:          1864          71304  <symbolKlass>
   8:          1247          70920  [Ljava.util.Hashtable$Entry;
   9:          2464          59136  com.ibm.edms.od.MyArrayList
  10:            34          54528  [B
  11:          1244          49760  java.util.Hashtable
  12:          1334          42688  java.util.LinkedHashMap$Entry
  13:           423          34344  <methodKlass>
  14:          1241          29784  java.util.ArrayList
  15:              16          24336  <constantPoolKlass>
  16:              15          15712  <constantPoolCacheKlass>
  17:              36          15328  [Ljava.util.HashMap$Entry;
  18:              38          12160  <objArrayKlassKlass>
  19:              16           8016  <instanceKlassKlass>
```

-heap parameter

The –`heap` parameter outputs the name and details of the garbage collector, the heap configuration, and the heap usage summary.

-permstat parameter

The –`permstat` parameter lists statistics for the objects in the permanent generation.

13.4.2 The jstat command

The **`jstat`** command provides performance and resource consumption information for an executing Java process. This utility is used to diagnose performance, heap sizing, and garbage collection issues. To see the options for your platform, enter **`jstat -help`** at the command line. Example 13-4 shows usage of the command in a Windows environment.

Example 13-4 jstat -help in a Windows environment

```
Usage: jstat -help|-options
       jstat -<option> [-t] [-h<lines>] <vmid> [<interval> [<count>]]

Definitions:
<option> An option reported by the -options option
<vmid> Virtual Machine Identifier. A vmid takes the following form:
                    <lvmid>[@<hostname>[:<port>]]
                Where <lvmid> is the local vm identifier for the target
                Java virtual machine, typically a process id;
<hostname> is the name of the host running the target Java virtual
machine; and
<port> is the port number for the rmiregistry on the target host. See
the jvmstat documentation for a more complete description of the
Virtual Machine Identifier.
<lines>       Number of samples between header lines.
<interval>    Sampling interval. The following forms are allowed:
                    <n>["ms"|"s"]
                Where <n> is an integer and the suffix specifies the
units as milliseconds("ms") or seconds("s"). The default units are
"ms".
<count>       Number of samples to take before terminating.
-J<flag>      Pass <flag> directly to the runtime system.
```

For more information about the **jstat** command, see the Sun Web page at the following address:

http://java.sun.com/j2se/1.5.0/docs/tooldocs/share/jstat.html

13.4.3 Heap Profiler (HPROF)

HPROF is a simple profiler agent that ships with JDK™ 5.0. The output of HPROF can be directed either to a file or a socket so that diagnostics can be performed locally or remotely.

The output of HPROF includes CPU usage, heap dumps, monitors, and threads. For a listing of supported HPROF parameters on your system, enter the following command:

```
java -agentlib:hprof=help
```

Example 13-5 shows the output of the **help** command in a Windows XP environment.

Example 13-5 Windows XP example of java -agentlib:hprof=help output

```
HPROF: Heap and CPU Profiling Agent (JVMTI Demonstration Code)

hprof usage: java -agentlib:hprof=[help]|[<option>=<value>, ...]

Option Name and Value  Description                 Default
---------------------  -----------                 -------
heap=dump|sites|all    heap profiling              all
cpu=samples|times|old  CPU usage                   off
monitor=y|n            monitor contention          n
format=a|b             text(txt) or binary output  a
file=<file>            write data to file
java.hprof[{.txt}]
net=<host>:<port>      send data over a socket     off
depth=<size>           stack trace depth           4
interval=<ms>          sample interval in ms       10
cutoff=<value>         output cutoff point         0.0001
lineno=y|n             line number in traces?      y
thread=y|n             thread in traces?           n
```

```
doe=y|n                 dump on exit?                 y
msa=y|n                 Solaris micro state accounting n
force=y|n               force output to <file>        y
verbose=y|n             print messages about dumps    y

Obsolete Options
----------------
gc_okay=y|n

Examples
--------
  - Get sample cpu information every 20 millisec, with a stack depth of
3:
      java -agentlib:hprof=cpu=samples,interval=20,depth=3 classname
  - Get heap usage information based on the allocation sites:
      java -agentlib:hprof=heap=sites classname

Notes
-----
  - The option format=b cannot be used with monitor=y.
  - The option format=b cannot be used with cpu=old|times.
  - Use of the -Xrunhprof interface can still be used, e.g.
       java -Xrunhprof:[help]|[<option>=<value>, ...]
    will behave exactly the same as:
       java -agentlib:hprof=[help]|[<option>=<value>, ...]

Warnings
--------
  - This is demonstration code for the JVMTI interface and use of BCI,
    it is not an official product or formal part of the JDK.
  - The -Xrunhprof interface will be removed in a future release.
  - The option format=b is considered experimental, this format may
change in a future release.
```

For more information about HPROF, see the Sun Developer Network Web page at the following address:

http://java.sun.com/developer/technicalArticles/Programming/HPROF.html

13.4.4 Java Heap Analysis Tool (jhat)

Starting with Java SE 6, Heap Analysis Tool (hat) has been replaced with jhat, which is included with the standard Sun distribution. jhat allows you to browse the objects in a heap snapshot generated from HPROF, and can identify all objects referenced by a root object.

For more information, see the Heap Analysis Tool 1.1 (HAT) page on java.net at the following address:

`https://hat.dev.java.net/`

Alternatively, see the jhat page at the following address on Sun's Web site:

`http://java.sun.com/javase/6/docs/technotes/tools/share/jhat.html`

13.4.5 Diagnostic Tool for Java Garbage Collector

The Diagnostic Tool for Java Garbage Collector is used to optimize Garbage Collector parameters when using the IBM JVM. It reads and analyzes the output of verbose garbage collection and produces text and graphical analysis output. The tool includes other advanced features such as the ability to analyze two or more files simultaneously.

Version 1.3 comes with built-in parsers for IBM JVM versions 1.5.0, 1.4.2, and 1.2.2. Developers can implement custom parsers for unsupported JVM levels.

For more information, see Diagnostic Tool for Java Garbage Collector page in alphaWorks at the following address:

`http://www.alphaworks.ibm.com/tech/gcdiag`

13.5 Testing tools

Several testing tools, as explained in the following sections, can help your Java development.

13.5.1 JUNIT

JUnit is a free unit-testing framework that supports the development of automated tests for the Java programming language. JUnit was created by Kent Beck and Erich Gamma and has become an important standard (as many would say) tool of test-driven development.

JUnit has been ported to many other languages, including PHP (PHPUnit), C# (NUnit), Python (PyUnit), Fortran (fUnit), Perl (Test:Class and Test:Unit), C++ (CPPUnit), and JavaScript (JSUnit).

For more information about JUNIT, see the Web page at the following address:

http://www.junit.org/home

13.5.2 ConTest

ConcurrentTesting - Advanced Testing for Multi-Threaded Applications (ConTest) improves the quality of testing by exposing potential problems earlier in the testing process. ConTest controls the scheduling and execution of program threads over multiple executions. Each time, it adds different sleep and yield instructions to exercise program thread synchronization. By using a preferences file, you can specify which of your classes are tested.

For more information, see the ConTest page at the following address:

http://www.alphaworks.ibm.com/tech/contest

13.5.3 Visual Performance Analyzer

Visual Performance Analyzer (VPA) is an Eclipse-based performance visualization toolkit. It includes the following major components:

- Profile Analyzer

  Profile Analyzer provides graphical and text-based views that allow for the analysis of performance problems to a particular process, thread, module, symbol, offset, instruction, or source line.

- Code Analyzer

  Code Analyzer examines executable files and displays detailed information about functions, basic blocks, and assembly instructions.

- Pipeline Analyzer

  Pipeline Analyzer displays the pipeline execution of instruction traces generated by an IBM POWER™ series processor.

- Counter Analyzer

  Counter Analyzer accepts hardware performance data from collection tools such as CPC or HPMCOUNT. The data is provided as XML and is parsed by this plug-in to allow visualizing and analysis.

- Trace Analyzer

  Trace Analyzer visualizes Cell Broadband Engine™ traces that contain information such as direct memory access (DMA) communication, locking and unlocking activities, and mailbox messages.
- Control Flow Analyzer

  Control Flow Analyzer is a tool that analyzes call trace data that is collected by such tools as Jprof, which is part of Performance Inspector. The call trace data contains information about each method call, such as how much time is spent in every invocation and who calls whom.

For more information, see Visual Performance Analyzer in alphaWorks at the following address:

http://www.alphaworks.ibm.com/tech/vpa

13.6 Getting support

You can seek help from the ODWEK user community or from the IBM Software Support team.

13.6.1 OnDemand User Group

The IBM DB2 Content Manager OnDemand User Group draws its membership from licensed OnDemand users. Their Web site features active forums that address many Content Manager OnDemand topics including the ODWEK Java APIs.

Visit the ODUG Web site at the following address:

http://odusergroup.org/

13.6.2 Opening a Problem Management Record

If you are unable to determine the root cause of a problem with the ODWEK Java APIs, you can open a Problem Management Record (PMR) with the IBM Software Support team.

To speed problem determination and avoid unnecessary delays, gather the arswww.trace file and other such diagnostic data as described in the MustGather ODWEK Java API documentation for your problem scenario. You can find the MustGather information at the following address:

http://www.ibm.com/support/docview.wss?rs=180&uid=swg21145599#generalMustGather

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see "How to get Redbooks" on page 277. Note that some of the documents referenced here may be available in softcopy only.

- *Content Manager OnDemand Guide*, SG24-6915
- *Implementing Content Manager OnDemand Solutions with Case Studies*, SG24-7511

Other publications

These publications are also relevant as further information sources:

- *DB2 Content Manager OnDemand for Multiplatforms Ver 8.4: Installation and Configuration Guide*, SC18-9232
- *DB2 Content Manager OnDemand: User's Guide*, SC27-0836
- *IBM Content Manager OnDemand for i5/OS Common Server ODWEK Installation and Configuration Guide*, SC27-1163
- *IBM DB2 Content Manager OnDemand for Multiplatforms Ver 8.4: Web Enablement Kit Implementation Guide*, SC18-9231
- *IBM Content Manager OnDemand: Messages and Codes*, SC27-1379

Online resources

These Web sites are also relevant as further information sources:

- IBM Content Manager OnDemand product site

  `http://www.ibm.com/software/data/ondemand`

  Go to the specific product page by selecting the product (either OnDemand for Multiplatforms, for i5/OS, or for z/OS and OS/390) and click **Go**. From the specific product page, you can access the following information:

  – Click the **Information Center** link to access the online information center.

  – Click the **Product documentation** link to obtain all manuals (in different languages) for the specific product.

  – Click the **Product support** link to get to Redbooks and Redpapers publications, technotes, and white papers.

  – Click other links such as **Demos**, **Developer resources**, and **Web casts** to obtain other information.

- Content Manager Ondemand V8.4 Information Center

  `http://publib.boulder.ibm.com/infocenter/cmod/v8r4m0/index.jsp`

- IBM Content Manager OnDemand User Group (ODUG)

  `http://odusergroup.org`

- IBM alphaWorks

  `http://www.alphaworks.ibm.com/`

  The Web site contains many tools mentioned in Chapter 13, “Troubleshooting” on page 251:

  – IBM Thread and Monitor Dump Analyzer

    `http://www.alphaworks.ibm.com/tech/jca`

  – HeapAnalyzer

    `http://www.alphaworks.ibm.com/tech/heapanalyzer`

  – HeapRoots

    `http://www.alphaworks.ibm.com/tech/heaproots`

  – Diagnostic Tool for Java Garbage Collector

    `http://www.alphaworks.ibm.com/tech/gcdiag`

– ConcurrentTesting - Advanced Testing for Multi-Threaded Applications (ConTest)

  http://www.alphaworks.ibm.com/tech/contest

– Visual Performance Analyzer (VPA)

  http://www.alphaworks.ibm.com/tech/vpa

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

IBM

Redbooks

IBM Content Manager OnDemand Web Enablement Kit Java APIs: The Basics and Beyond

(0.5" spine)
0.475"<->0.875"
250 <-> 459 pages

IBM Content Manager OnDemand Web Enablement Kit Java APIs

The Basics and Beyond

Develop Web applications by using ODWEK V8.4 Java APIs

Gain insightful best practices, hints, and tips

Tune and troubleshoot OnDemand Web applications

IBM Content Manager OnDemand is the industry leading report management product. It provides enterprise report management and electronic statement presentment. It is high-performance middleware for automatic management of formatted computer output and reports. It helps companies gain significant return on investment by transforming costly high-volume print output to electronic information capture and presentation in support of customer service.

In this IBM Redbooks publication, we provide an overview of the OnDemand Web Enablement Kit (ODWEK) version 8.4 Java APIs and explain the commonly used APIs for application development. In addition, we examine the capabilities and usage of the APIs through use cases, best practices, hints and tips, and code snippets. We explain connection pooling, folder searching, document retrieval, document storing and updating, memory and performance, and troubleshooting in terms of application development.

ODWEK Java APIs can be incorporated into any Java-based application, including stand-alone applications, portlets, servlets and Web services. We illustrate the APIs by using servlet-based code.

This book is intended for application developers who are responsible for developing Web applications that interface with Content Manager OnDemand. It also serves as a good reference guide for developers and system administrators to fine-tune and troubleshoot Content Manager OnDemand Web applications.

SG24-7646-00 ISBN 0738431761